SEARCHING BUSINESS PROCESS MODELS BY EXAMPLE

MATTHIAS KUNZE

BUSINESS PROCESS TECHNOLOGY
HASSO PLATTNER INSTITUTE, UNIVERSITY OF POTSDAM

DISSERTATION
ZUR ERLANGUNG DES AKADEMISCHEN GRADES EINES
DOKTORS DER NATURWISSENSCHAFTEN
– DR. RER. NAT. –

June 2013

Published online at the
Institutional Repository of the University of Potsdam:
URL http://opus.kobv.de/ubp/volltexte/2013/6884/
URN urn:nbn:de:kobv:517-opus-68844
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-68844

Matthias Kunze: *Searching Business Process Models by Example,* eingereicht an der Mathematisch-Naturwissenschaftlichen Fakultät der Universität Potsdam. Juni 2013.

To myself I am only a child playing on the beach,
while vast oceans of truth lie undiscovered before me.

— Isaac Newton

ABSTRACT

Business processes are fundamental to the operations of a company. Each product manufactured and every service provided is the result of a series of actions that constitute a business process. Business process management is an organizational principle that makes the processes of a company explicit and offers capabilities to implement procedures, control their execution, analyze their performance, and improve them. Therefore, business processes are documented as process models that capture these actions and their execution ordering, and make them accessible to stakeholders. As these models are an essential knowledge asset, they need to be managed effectively. In particular, the discovery and reuse of existing knowledge becomes challenging in the light of companies maintaining hundreds and thousands of process models. In practice, searching process models has been solved only superficially by means of free-text search of process names and their descriptions. Scientific contributions are limited in their scope, as they either present measures for process similarity or elaborate on query languages to search for particular aspects. However, they fall short in addressing efficient search, the presentation of search results, and the support to reuse discovered models.

This thesis presents a novel search method, where a query is expressed by an exemplary business process model that describes the behavior of a possible answer. This method builds upon a formal framework that captures and compares the behavior of process models by the execution ordering of actions. The framework contributes a conceptual notion of behavioral distance that quantifies commonalities and differences of a pair of process models, and enables process model search. Based on behavioral distances, a set of measures is proposed that evaluate the quality of a particular search result to guide the user in assessing the returned matches. A projection of behavioral aspects to a process model enables highlighting relevant fragments that led to a match and facilitates its reuse. The thesis further elaborates on two search techniques that provide concrete behavioral distance functions as an instantiation of the formal framework. Querying enables search with a notion of behavioral inclusion with regard to the query. In contrast, similarity search obtains process models that are similar to a query, even if the query is not precisely matched. For both techniques, indexes are presented that enable efficient search. Methods to evaluate the quality and performance of process model search are introduced and applied to the techniques of this thesis. They show good results with regard to human assessment and scalability in a practical setting.

ZUSAMMENFASSUNG

Geschäftsprozesse bilden die Grundlage eines jeden Unternehmens, da jedes Produkt und jede Dienstleistung das Ergebnis einer Reihe von Arbeitsschritten sind, deren Ablauf einen Geschäftsprozess darstellen. Das Geschäftsprozessmanagement rückt diese Prozesse ins Zentrum der Betrachtung und stellt Methoden bereit, um Prozesse umzusetzen, abzuwickeln und, basierend auf einer Auswertung ihrer Ausführung, zu verbessern. Zu diesem Zweck werden Geschäftsprozesse in Form von Prozessmodellen dokumentiert, welche die auszuführenden Arbeitsschritte und ihre Ausführungsbeziehungen erfassen und damit eine wesentliche Grundlage des Geschäftsprozessmanagements bilden. Um dieses Wissen verwerten zu können, muss es gut organisiert und leicht auffindbar sein – eine schwierige Aufgabe angesichts hunderter bzw. tausender Prozessmodelle, welche moderne Unternehmen unterhalten. In der Praxis haben sich bisher lediglich einfache Suchmethoden etabliert, zum Beispiel Freitextsuche in Prozessbeschreibungen. Wissenschaftliche Ansätze hingegen betrachten Ähnlichkeitsmaße und Anfragesprachen für Prozessmodelle, vernachlässigen dabei aber Maßnahmen zur effizienten Suche, sowie die verständliche Wiedergabe eines Suchergebnisses und Hilfestellungen für dessen Verwendung.

Diese Dissertation stellt einen neuen Ansatz für die Prozessmodellsuche vor, wobei statt einer Anfragesprache Prozessmodelle zur Formulierung einer Anfrage verwendet werden, welche exemplarisch das Verhalten der gesuchten Prozesse beschreiben. Dieser Ansatz fußt auf einem formalen Framework, welches ein konzeptionelles Distanzmaß zur Bewertung gemeinsamen Verhaltens zweier Geschäftsprozesse definiert und die Grundlage zur Suche bildet. Darauf aufbauend werden Qualitätsmaße vorgestellt, die einem Benutzer bei der Bewertung von Suchergebnissen behilflich sind. Verhaltensausschnitte, die zur Aufnahme in das Suchergebnis geführt haben, können im Prozessmodell hervorgehoben werden. Die Arbeit führt zwei Suchtechniken ein, die konkrete Distanzmaße einsetzen, um Prozesse zu suchen, die das Verhalten einer Anfrage exakt enthalten (Querying), oder diesem in Bezug auf das Verhalten ähnlich sind (Similarity Search). Für beide Techniken werden Indexstrukturen vorgestellt, die effizientes Suchen ermöglichen. Abschließend werden allgemeine Methoden zur Evaluierung von Prozessmodellsuchansätzen vorgestellt, mit welchen die genannten Suchtechniken überprüft werden. Im Ergebnis zeigen diese eine hohe Qualität der Suchergebnisse hinsichtlich einer Vergleichsstudie mit Prozessexperten, sowie gute Skalierbarkeit für große Prozessmodellsammlungen.

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

- Matthias Kunze and Mathias Weske. Metric Trees for Efficient Similarity Search in Process Model Repositories. In *Business Process Management Workshops*, volume 66 of *Lecture Notes in Business Information Processing*, pages 535–546. Springer, September 2010.

- Matthias Kunze, Matthias Weidlich, and Mathias Weske. m^3 — A Behavioral Similarity Metric for Business Processes. In *Central-European Workshop on Services and their Composition*, volume 705, pages 89–95. ceur-ws.org, Feb 2011.

- Matthias Kunze, Matthias Weidlich, and Mathias Weske. Behavioral Similarity — A Proper Metric. In *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2011.

- Ahmed Awad, Sherif Sakr, Matthias Kunze, and Mathias Weske. Design by Selection: A Reuse-Based Approach for Business Process Modeling. In *Conceptual Modeling (ER)*, volume 6998 of *Lecture Notes in Computer Science*, pages 332–345. Springer, 2011.

- Matthias Kunze, Alexander Luebbe, Matthias Weidlich, and Mathias Weske. Towards Understanding Process Modeling — The Case of the BPM Academic Initiative. In *Business Process Model and Notation*, volume 95 of *Lecture Notes in Business Information Processing*, pages 44–58. Springer, 2011.

- Matthias Kunze and Mathias Weske. Local Behavior Similarity. In *Enterprise, Business-Process and Information Systems Modeling*, volume 113 of *Lecture Notes in Business Information Processing*, pages 107–120. Springer, 2012.

- Markus Guentert, Matthias Kunze, and Mathias Weske. Evaluation Measures for Similarity Search Results in Process Model Repositories. In *Conceptual Modeling (ER)*, volume 7532 of *Lecture Notes in Computer Science*, pages 214–227. Springer, 2012.

- Sherif Sakr, Ahmed Awad, and Matthias Kunze. Querying Process Models Repositories by Aggregated Graph Search. In *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 573–585. Springer, 2012.

- Remco M. Dijkman, Boudewijn F. van Dongen, Luciano Garcia-Banuelos, Marlon Dumas, Matthias Kunze, Henrik Leopold, Jan Mendling, Reina Uba, Matthias Weidlich, Mathias Weske, and Zhiqiang Yan. A Short Survey on Process Model Similarity. In *Seminal Contributions to Information Systems Engineering*, pages 421–427. Springer, 2013.
- Matthias Kunze and Mathias Weske. Methods for Evaluating Process Model Search. In *Business Process Management Workshops*, *Lecture Notes in Business Information Processing* (to appear). Springer, 2011.
- Matthias Kunze and Mathias Weske. Visualization of Successor Relations in Business Process Models. In *Web Services and Formal Methods*, *Lecture Notes in Computer Science* (to appear). Springer, 2013.
- Matthias Kunze, Matthias Weidlich, and Mathias Weske. Querying Process Models by Behavior Inclusion. In *Software and Systems Modeling*, (to appear). Springer, 2013.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

BACKGROUND

1 INTRODUCTION

Why worry about business processes? This question can be answered by asking another. How do companies do their work and why do they do it that way? Every business is process-driven. Business processes are the very substance of organizing a company, as they constitute the connection between pieces of work conducted by individuals. Business process models capture how operations are carried out and make this knowledge explicit and accessible to all stakeholders. Since process models are an essential asset of every business, companies have increasingly documented their process knowledge. Yet, with more and more processes being modeled, the individual knowledge vanishes in the sheer plentitude of information.

In our research, we face the challenge of rediscovering the individual knowledge hidden in large process model collections and making it available for reuse. To this end, we propose a novel approach to searching business process models that assumes a most process-oriented representation of a search question — a business process model that serves as an example of the desired response. The present thesis reports on our research and the achieved results.

This chapter outlines the content of this thesis. After a brief investigation of the drivers for searching business process models in Sect. 1.1, we discuss the objective of our research in Sect. 1.2, and the contributions made in Sect. 1.3. Section 1.4 concludes with an outline of this thesis and provides some advice on how to read it.

1.1 DRIVERS FOR PROCESS MODEL SEARCH

Business process models are a cornerstone of modern organizations as they explicitly capture the knowledge to carry out the operations of a business and are used for documentation, analysis, automation, monitoring, and certification, among others. Each product an organization manufactures and every service it offers are the result of a series of actions that are performed by humans or automatically in a coordinated fashion that respects the dependencies between these actions. The actions and their dependencies comprise a business process. Essentially, business processes are reassembling the work that has been broken into standardized and routinized tasks at the beginning of the twentieth century. Making explicit the connections between individual departments and functional divisions, business process models make the work of companies transparent and enable its improvement with regard to the overall process.

Hammer and Champy [114] and Davenport [63] have recognized the potential of business processes two decades ago and held out for a radical change in the way companies organize their work. This change has taken place [266] and in recent years, companies have more and more captured their operations, aligned them with their business strategies, and documented them in terms of business process models. Nowadays, these companies maintain hundreds and thousands of business process models as a vital knowledge asset [77]. However, the sheer number of process models that have been modeled by individuals, originated from different sources and kept in isolated places, raises manifold challenges. To be of value, these models need to be consulted, analyzed, revised, or incorporated into other models — in short, they are reused in various ways.

Process model repositories have been proposed as a solution to manage business process models in an integrated fashion, see, for instance, [49, 163, 331]. A repository is a "shared database of [...] artifacts produced or used by an enterprise" [31] and provides features for the storage, retrieval, manipulation, and lifecycle management of these artifacts. Business process model repositories comprise specific functions with regard to process models, e. g., for their analysis or comparison, for refactoring, and to manage sets of process variants.

Process model search is an essential feature of business process repositories that is driven by the need to discover and harness the knowledge concealed in vast process model collections. We highlight a number of situations that make use of business process model search.

PROCESS MODELING Business process modeling is a tedious and time-consuming effort. However, before creating new process models from scratch, users may search for models that have been created already and represent the business process to be

documented — or at least parts thereof. Search must enable users to express their information needs and retrieve process models to reuse and revise them to fit their needs. This suggests saving time and reducing errors during process modeling, and hence, increases the quality of new process models.

MODEL MANAGEMENT The management of large collections of process models that are created by different individuals can lead to redundancies among them. On the one hand, a model collection may contain duplicates or variants of process models that need to be harmonized. Some process models may overlap in their described functionality or identical functionality may be described in different ways, which can be resolved by means of process model refactoring. On the other hand, the compliance of process models with business or legal constraints must be ensured, as these models are the blueprint to carry out a business' operations. Hence, capabilities to identify similar process models, to discover duplicate fragments, and to check the compliance of business processes are needed.

PROCESS ENACTMENT Search can also support the execution of business processes. For example, specific cases from a large set of process instances can be revealed by searching with a process model that describes desired properties.

Business process models can be derived from a number of execution logs of historic cases of that process. Here, reference models for those derived from a log can be obtained from a process model repository.

The drivers mentioned above are by no means comprehensive, but they illustrate the variety of situations that require process model search or can benefit from it significantly. Despite the diversity of these drivers, we can identify two fundamental methods to search for business process models. *Querying* allows the retrieval of process models of which only few, yet important features are known a priori that must be comprised by any match. For instance, a process designer may search for existing process models by a small set of actions and their dependencies, rather than with a complete process model. Compliance checking also incorporates queries, i. e., a description of few, yet specific rules that need to be precisely fulfilled by business process models. *Similarity search*, in contrast, is based on the comparison of a pair of complete process models and evaluates how much they resemble each other, including cases where a process model does not precisely match a query. This kind of search provides a fruitful solution to the discovery of variants and duplicates, or to find reference process models for those revealed by process mining.

Search does not only comprise the process of satisfying some information need by finding process models that answer a particular

search question, but concerns the collection and organization of information required to provide an answer efficiently. That is, without examining each model stored in the repository with regard to its fulfillment of the search question. Search also deals with the presentation of a ranked search result, such that the most relevant answers are presented first and in a way that facilitates their reuse.

1.2 RESEARCH OBJECTIVE

Research in the area of business process management has recognized the need for process model search and its potential for process model management. In a former study [131], the management of process models has been identified as one of the top ten challenges of business process modeling for research and industry. Consequently, there exists a large body of work that addressed similarity measures and approaches to process model querying, which we examine in the related work sections of this thesis.

State of the art

The majority of solutions proposed to similarity search and querying focuses only on the semantics of quantifying the similarity or distance of a pair of similar process models or to decide a match by means of a set of features, respectively. Searching among a collection of process models then requires comparing a query with each model in the collection and assessing its relevance for a match. Even if the implementation of this assessment shows good performance, such an approach cannot be considered efficient and does not scale well with rapidly growing process model repositories.

Moreover, the presentation of search results is largely neglected. A search result needs to be ranked to present relevant matches first and indicate the degree how well a proposed answer satisfies the search question. The commonalities, by which relevance has been assessed, should be visualized to facilitate comprehension of an answer. For similarity search, a ranking can be easily obtained by the similarity of a model to the query, whereas related work on querying generally resorts to deciding a match, i. e., answering whether a model matches the query or not. The latter issue, i. e., facilitation of comprehension and reuse, remains largely unsolved, yet.

Defined as a series of actions that are carried out in a coordinated fashion, the primary focus of business process models is on the description of behavior. Therefore, we argue that search should also assume a behavioral perspective of the way how process models are compared. A number of approaches for similarity search and querying has been proposed in literature, but no uniform solution has been presented that allows for both search methods by the same primitives. Querying for process model behavior, in particular, has been tackled by means of expressive, yet intricate query languages. Non-expert users are unlikely to employ such a means for search.

Research goals

These drawbacks of existing solutions restrict their applicability in practice. Drawing from the drivers for process model search, we propose a solution that challenges these restrictions in a consistent manner and enables a broad spectrum of users to search for process models. Therefore, we formulate the research objective of this thesis as follows.

> *Design and evaluate a formal framework to search for process models by providing an example that describes the behavior of a desired answer.*
>
> *This framework shall enable efficient querying and similarity search, and facilitate reuse by presenting results in a uniform and comprehensible manner.*

We elaborate on the research objective and determine a set of goals that shall be met on the journey throughout this thesis to achieve the above objective.

1. *Establish a search framework that enables querying and similarity search likewise.*
   Following from the diversity of drivers for process model search, we aim at a conceptual framework for searching process models that meets the aforementioned requirements, i. e., enables the organization of information to answer queries efficiently and provides a common presentation of search results. Techniques to facilitate the comprehension of matches and their reuse shall be developed.

2. *Employ a specification of the search question that is suited for users who are capable of understanding and formulating business process models.*
   We argue that process model search shall be generally applicable by users that handle business process models on a regular basis including, in particular, stakeholders that are familiar with process modeling yet are no experts. Hence, the specification of a search question must find a well-balanced tradeoff between its expressiveness, simplicity, and effort required to formulate the question.

3. *Design techniques for querying and similarity search that focus on the behavior of business processes.*
   Stemming from the conceptual framework and the specification of a search question, techniques to search for process models by querying or similarity shall be devised that operate on the same primitives and fit with the framework. As process models describe actions and their dependencies, these techniques shall focus on behavior expressed by process models.

4. *Evaluate these techniques with regard to their quality and performance.*
   Judging on the applicability and usefulness of the proposed framework and search techniques requires comprehensive evaluation. Quality refers to the relevance of proposed search results with regard to human assessment, whereas performance applies to the techniques' efficiency and scalability in large process model collections.

Foundations

These goals enclose the scope of this thesis. To fulfill them, we rely on a number of concepts and achievements that have been presented before. We briefly portray these concepts, as they are not our contribution. A more elaborate discussion of them can be found in Chap. 3.

An essential preliminary to compare business process models is the construction of a *process alignment*, that is, relevant features of one process model need to be mapped to corresponding features in another process model. Based on these correspondences, it becomes possible to examine the relations between the features of one process model with relations between the aligned features of the other. Since process models principally describe the operations of a business, these features are typically the actions of a process model. Process alignments must cope with different kinds of heterogeneity, as it is unlikely that actions of process models that have been designed by different individuals use identical terms, and the construction of an alignment is far from trivial. A large body of work has been devoted to the topic, and a framework has been presented in [316].

Petri nets have been proposed as a common formalism for the behavior of business process models. Their syntax and semantics are founded on a strong mathematical foundation [283]. Research dedicated to Petri nets has provided an abundance of important classes and properties have been shown that we rely on for our formalization.

Successor relations are an abstraction of the behavior of business processes that capture the potential execution order of pairs of actions. Informally, if two actions can be executed after one another, they are in a successor relation. Such relations have been introduced, e. g., for process mining [195], matching web services [84], and consistency management [315]. Recently, a set of enhanced relations has been proposed [314] that capture behavioral relations between actions more distinctive. Successor relations capture the behavior in a finite and compact representation, which makes them favorable candidates for process model search.

1.3 CONTRIBUTIONS

Building on the above foundations, this thesis contributes a novel approach to searching for process models with behavioral semantics that takes an example process model as input and enables querying

and similarity search. Querying reveals all process models for which the example process represents a precise behavioral abstraction, or informally, that allow replaying the behavior expressed in the query. Similarity search finds process models that resemble the query to some extent with regard to common successor relations and therefore show a behavioral overlap.

Parts of this thesis have been published before, cf. [153, 159, 158, 12, 157, 154, 110, 254, 78, 155, 156, 160]. This thesis summarizes results of these works and extends them. In the following, we explain the contributions made by this thesis in more detail.

Searching Process Models by Example

Search by example denotes an approach that takes a regular process model as input and searches for regular process models in a process repository, whereas comparison of the behavior of process models is independent of a particular process modeling language. Our approach resorts to successor relations, an abstraction of the behavior of process models that is computed from the traces a process can provide and therefore applies to any process specification. For common process formalisms, we explain how behavioral commonalities of business processes are discovered from successor relations, and how distances can be computed that quantify the compliance of a process model with a query. We discuss properties of such distances that provide the grounding for searching process models and presenting search results. This framework founds the basis of concrete techniques for querying and similarity search, and is concerned with the first and second of our research goals.

Process Model Querying

We contribute process model querying as an instantiation of the aforementioned framework, where the query — a regular process model — expresses behavioral constraints that must be met by any process model of the search results. A match is decided by a behavioral abstraction, called abstract trace inclusion, that requires that each trace a query can produce must be mirrored and may be extended by a business process. We show, for sound, free-choice workflow systems that this abstraction can be decided purely by successor relations. The amount of abstraction required for abstract trace inclusion then yields a distance that is used for ranking. Efficient search is enabled by means of an inverted index. A necessary condition that is computed efficiently allows excluding models from search that cannot match a query. Process model querying contributes the first part to our third research goal.

Process Model Similarity Search

Similarity search evaluates how much two process models resemble each other and provides a set of process models most similar to a given query as search result. As second contribution to our third research goal, we present a notion of behavioral similarity that measures commonalities in the successor relations of two process models and incorporates the confidence of an alignment, i. e., how well pairs of corresponding actions fit together. Relation sets, which distinguish exclusiveness, strict order, and interleaving order in the execution of actions, are used to construct an enhanced and configurable similarity measure. Therefore, we introduce a method to automatically configure this measure in the absence of a training set. Again, we show how search can be carried out efficiently, by means of a metric space index.

Presentation of Search Results

Comprehension of how a search result came about is crucial for effective search, particularly, if the match is not obvious from looking at the process models. Hence, once a search question has been posed and matches of the search result have been decided, they need to be presented to the user in a ranked order such that the most relevant matches are presented first. Following our second research goal, to address a broad audience of users, search results need to be enriched with information that facilitates users comprehending the proposed matches. On the one hand, we introduce a set of quality measures based on our notion of distances that give insights into the confidence and discrimination of a search result, and are able to improve its ranking. On the other hand, those parts of a process model that contributed to a match should be visualized in order to help the user understanding why a certain result has been proposed. Our contribution with that regard is a projection of successor relations, which describe dynamic behavior, to the static structure of a process model's graph.

Evaluation Methods for Process Model Search

The last of our research goals refers to the comprehensive evaluation of the proposed techniques. From surveying related work, we have observed that there exists no agreed corpus of measures and methods for process model search. Therefore, we develop a methodology that incorporates quality measures from information retrieval and applies them specifically to process model search. We distinguish between quality and performance. Quality refers to the relevance of identified matches, their ranking, and the robustness of a technique to certain factors. In comparison, performance concerns consumption of resources, efficiency, and scalability of a process model search tech-

nique. Besides measures for these categories, we present versatile methods to analyze, visualize and, explain evaluation results. These methods are used to evaluate querying and similarity search techniques proposed in this thesis. Additionally, we examine the effectiveness of the aforementioned quality measures for search results.

1.4 STRUCTURE OF THE THESIS

We conclude this chapter with an outline of this thesis and some advice on how to read this document. Figure 1 summarizes the concepts mentioned above and their relationship, which we briefly explain in the following.

Part I: Background

The first part of this thesis paves the way for the presentation of the formal framework to searching process models by example and contains no original contributions of the thesis. Chapter 2 reports on the background of business process management, an organizational principle based on the design and analysis, implementation, and enactment of business process models, as well as the evaluation of process performance, cf. Fig. 1. We explain how process models are managed and motivate the need for powerful capabilities to search process models in order to effectively reuse the preserved knowledge. A comprehensive discussion on the background and challenges of process model search that yields a lifecycle on its own founds the basis for our contributions.

In Chap. 3, we briefly recall basic notations and introduce formal preliminaries this thesis relies on, including Petri nets, successor relations, and process alignments. We elaborate on the groundwork

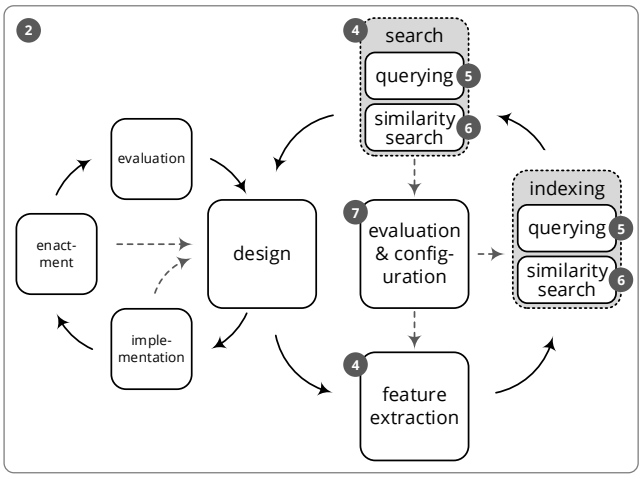


Figure 1: Overview of the thesis. Numbers represent chapters that address respective topics. Chapter 2 elaborates on this overview in detail.

of the latter and means to construct process alignments in a concise fashion.

Readers that are only interested in the technical contributions made in this thesis can safely skip this part. In particular, Chap. 3 can be omitted by readers that are already familiar with the mentioned concepts, as we will refer to particular definitions when they are required throughout this thesis.

Part II: Searching Process Models by Example

Answers to our research objective and the majority of contributions, mentioned in the previous section, are found in the second part of this thesis. Chapter 4 introduces the formal framework to search for business process models by example, thereby emphasizing commonalities that can be identified by means of successor relations and how these features are extracted from process models, see Fig. 1.

We discuss the construction and properties of distances that enable process model search. Before introducing particular techniques for querying and similarity search, we introduce measures to predict the quality of a search result, and a means to highlight parts of a matching model that are relevant with respect to a query by projecting successor relations on the model graph.

Subsequently, we introduce process model querying in Chap. 5 and similarity search in Chap. 6 as concrete instantiations of the formal framework. For both techniques, we explain how a match is decided and a distance function derived that guides the ranking of search results. To facilitate these techniques in practice, we present index structures that avoid examining every process model of a collection and thereby make search efficient.

Part III: Evaluation and Discussion

Following our research objective, we evaluate the presented techniques in Chap. 7. Evaluation provides an assessment of the quality and performance of a search technique and fuels their improvement by determining optimal configurations as depicted in Fig. 1. Therefore, we first introduce a methodology to judge on the quality and performance of process model search techniques and make different techniques comparable. Subsequently, we apply this methodology to the presented search techniques, querying and similarity search, and evaluate them accordingly.

Chapter 8 concludes this thesis. We first report on the implementation of a search platform, which has been devised as a manifestation of the search framework, and dedicated search engines for querying and similarity search. Following, we reflect on the results achieved in this thesis concerning the stated research objective. To conclude our

work, we examine these results with regard to their assumptions and limitations and elaborate on venues for future work on this topic.

Reading Advice

Our research focuses on the discovery of individual knowledge from a large body of information in an efficient and effective fashion. Consequently, we added some features to this document that aim at supporting the reader in the discovery of individual knowledge and findings in this thesis.

- Each chapter of this thesis is prepended with a brief abstract that relates the content of the respective chapter with the overall context of process model search. This is intended to assist the reader in locating relevant information fast and skipping sections that are of less interest to them.
- To emphasize important notions, hypotheses, or findings, we added margin notes to the document. They too, are intended to guide the reader in locating relevant material.
- Each chapter of Part II introduces a running example by which methods and results are explained in a coherent fashion. Appendix B of this thesis contains a summary of figures and tables related to the running example of each chapter. They have been devised to be referred to during the lecture of the respective chapter. In the printed version of this document, these summaries can be folded out.
- Moreover, to guide the reader through the wealth of symbols introduced for relations, sequences, formulae, etc., a glossary is added to Appendix C
- An overview of the related work discussed for querying and similarity search in the context of business process models is provided in Appendix D.

2 MANAGEMENT OF BUSINESS PROCESS MODELS

Business process management has received increasing recognition in the last two decades and led to a paradigm shift in the way businesses are run. Vital to managing an organization, business processes sustain a company's competitiveness and flexibility to react to changing market conditions. Consequently, these companies maintain hundreds and thousands of business process models that streamline their working procedures and facilitate process implementation. Yet, the management of large collections of business process models requires powerful capabilities to search and reuse the preserved knowledge.

In this chapter, we report on the background of business process management and the foundations of a process-oriented perspective of running a business in Sect. 2.1. The basics of business process models and their role as key enabler for effective business process management are elucidated in Sect. 2.2, before we briefly turn our attention to process model repositories that provide features to store and manage business process models and related artifacts, cf. Sect. 2.3.

Finally, we examine the organizational and technical foundations of process model search in general, covering different dimensions of search, search methods, and a lifecycle of process model search in Sect. 2.4. We conclude the chapter with a discussion of challenges and requirements that need to be addressed by any comprehensive search technique.

2.1 BUSINESS PROCESS MANAGEMENT

> *Business Process Management is a comprehensive system for managing and transforming organizational operations, based on what is arguably the first set of new ideas about organizational performance since the Industrial Revolution.*
>
> Michael Hammer [113]

Not a single business exists that operates without business processes. Defined as a series of actions or changes that are conducted to achieve a particular goal, processes exist everywhere. Business processes are fundamental to the operations of a business that consist of observable actions to manufacture products and serve customers, and to reward these efforts by increased revenue. However, processes may not always be apparent.

Literature offers various definitions for business processes, e. g., "a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer" [114], "a structured, measured set of activities designed to produce a specified output for a particular customer" [63], "the complete and dynamically coordinated set of collaborative and transactional activities that deliver value to customers" [266], and many more, see [288, 91, 322, 90, 80]. All these definitions have in common that they focus on the actions that are conducted to transform some input into some output that is targeted at the business' customer. On the basis of [322], we define the term business process as follows.

Definition 2.1 (Business Process)**.**
A *business process* consists of a series of actions that are conducted in a coordinated fashion to jointly realize a goal that is of value to a particular customer. A business process is embedded into an organizational, informational, and technological environment. ◂

Business process management (BPM) makes the processes of a business explicit by offering a holistic picture of how organizations create value, which actions they carry out, how these are interrelated, and how they are embedded in their environment. To this end, BPM provides solutions on a spectrum from economical and administrative issues of a business, such as best practices to establishing a process-oriented perspective, to technical implementations, for instance the automatic enactment of business processes using workflow technology. By providing an integration layer that makes the processes of a company visible among the complete spectrum, business process management strives to close the so-called "Business IT Gap" [107].

Levels of control

Business processes can be defined on various levels of control, i. e., strategic, organizational or tactical, operational, and implemented business processes [243, 322], that span the aforementioned spectrum

covered by business process management. Strategic processes typically refer to business goals that are further refined in organizational business processes that focus on the interdependencies of various business processes, their input, and their output. Operational processes are more consistent with Def. 2.1, as they are concerned with the actions that are undertaken to achieve a particular goal and their temporal and causal relationships. Implemented business processes are enriched with technical and organizational details toward the realization of a business process in its environment.

Different levels of control also yield different levels of abstraction of the business processes. This is due to the fact that, the higher the level of control, the more processes need to be managed as a coherent unit. For instance, to steer a company, all processes of a business must be considered at once, which requires reducing the detail of each process description to make this task manageable. Operational processes, on the contrary, need to be sufficiently detailed to describe the actions carried out and their dependencies unambiguously. However, all business processes within a level of control should comprise the same level of abstraction to provide for consistency among them. In some cases, implemented business processes have an explicit form, i. e., a software artifact or configured web services, however the majority of business processes is still carried out manually [113]. Therefore, we resort to operational business processes hereafter.

The environmental factors that impact business processes — a company's organization, the information required for conducting work, and the technical infrastructure, e. g., information systems and manufacturing equipment — are, at the same time, agreed to be the key enablers of business process management.

Definition 2.2 (Business Process Management)**.**
Business process management comprises methods, techniques, and systems to design and analyze, implement, enact, and evaluate operational business processes involving organizations, information, and technological infrastructure. ◂

The evolution of BPM

Two decades ago, Davenport [63] and Hammer and Champy [114] called for a fundamental change in the way how organizations structure and manage their work. The demand for radical change to save an organization may be questioned, but beside their repeated call for drastic reengineering efforts, the authors describe the essence of business process management and process-oriented organizations. Within the last decade, the stipulated change from functional silos to process-orientation has taken place [266]. Business process management has experienced remarkable progress which becomes apparent on miscellaneous grounds. For instance, the ARIS approach to enterprise modeling [257] that focuses on a holistic process-oriented perspective on the administration of business has long dominated

industry in business process management. The tremendous effort to standardize BPMN [225], a process modeling language that "is readily understandable by all business users [and] creates a standardized bridge for the gap between the business process design and process implementation" [225], joined market competitors. Process automation, a topic that dates back over 40 years [289], has finally been acknowledged by large BPM software vendors who strive to support BPMN-compliant process engines. We have even seen business process management methods being applied in healthcare, e. g., [119], to support the treatment of hospital patients.

Precursors of BPM

As it turns out, methods of business process management can be traced back to early attempts to improve the operational performance of businesses. Business process management has its origins in business administration [91, 322], quality management, systems engineering, and IT innovation [63], and is an organizational principle that is influenced by the way to efficiently organize and integrate the resources of an organization — its workforce, the means of production, and technology — along the dimension of business processes. Following the ideas of systems engineering to structure the operations of an organization based on the principles of rationality and efficiency, complex work is decomposed into measurable, routinized procedures that are carried out in coordination. The efficiency of the overall process can then be assessed by aggregating procedural measures. This allows for the continuous measurement of quality and performance, and thereby, for responding to changes within an organization or its environment in an agile and effective fashion.

With regard to quality management, business process management aligns the operations of an organization with the desired outcome from the perspective of the customer, that is, it strives to minimize quality variations and defects. At the same time, all actions conducted shall provide an output that is of value to the customer of this process [114]. Information technology can spark dramatic change and improvement over the status quo and is denoted one of the central enablers for business process management [63, 114, 288]. For instance, the utilization of a central information system provides impromptu insight into real time data and reduces replication and synthesis of various documents into a consistent case, e. g., an order document, a delivery note, and an invoice for a single order.

A Process-Oriented Perspective

Traditionally, companies are organized by a functional perspective, i. e., a set of actions are performed within a well-defined and isolated scope of responsibility. These functions are associated with primary and secondary activities of Porter's renowned value chain model [238], e. g., logistics, operations, and marketing, and are organized in respective departments. Each department is responsible

only for the functions that it is in control of and improvement is typically limited to the boundaries of departments. Business processes are cross-functional, i. e., they create the operational glue between these departments. In process-oriented organizations the notion of a business process is made explicit and dominates the organization of resources.

For an example, consider the order fulfillment process depicted in Fig. 2 that has been described in [114]. An order is received by customer service and verified for its formal correctness. Afterwards, inventory management checks, whether the ordered goods are in stock and back-orders them from a manufacturer if they are not. Once the goods are available, they can be packed and their shipment to the customer be issued by product handling. Subsequently, finance can prepare an invoice, send it to the customer, and ensure its payment in due time. The customer's order is the input to the process and the products delivered to and paid by the customer its output.

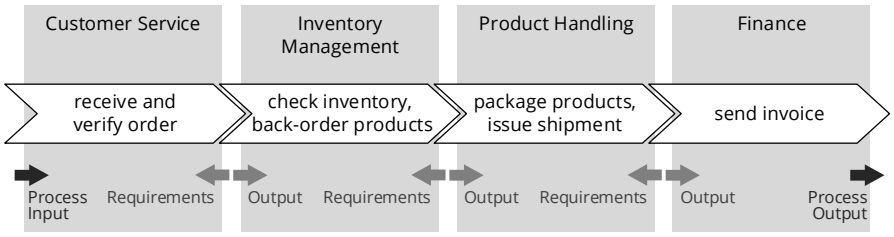


Figure 2: An exemplary order fulfillment process crosses functional departments.

Process-orientation requires that all operations are aligned with the underlying business process and are tailored toward improving its output [114]. Responsibilities are not assigned to the functional department, but aligned with each case, e. g., a process representative is responsible for an order from its reception to its fulfillment. From this, the benefits of process-orientation become clear. As the process connects individual parts of a business that are manifested in the underlying functions, the interfaces between departments are made explicit in the context of the business process. Each subsequent department is a customer of the results of the previous department that becomes a provider. This allows the precise definition of requirements toward the output, i. e., goods and information that are handed over from the provider to the customer [91, 322]. In turn, this enables reducing goods and services delivered from one department to another that do not contribute to the overall process result, and thereby improve the business process.

Benefits of process-orientation

A process-oriented perspective makes the operations of a business transparent, in that dependencies, costs and benefits, and required resources become apparent. Also, it enables tracking the state of each case across the functional departments and responding quickly to customer inquiries [114]. Statistical information of historic cases can

be used to make predictions about future states and termination of a running process. Operations carried out within the boundaries of a business process can be evaluated with respect to their cost, e. g., material and time, and their benefits, i. e., their contribution to the customer value of the process output, which is essential for business process improvement [114, 91, 322].

Finally, process-orientation sustains a company's competitiveness and flexibility to react to changing conditions in their environment, e. g., market demands and legal regulations. Improvement of operations within functional departments has only limited impact on the improvement of the overall business. If, for instance, the inventory management in the above example was improved tenfold, the company would still not be able to handle significantly more incoming orders, as the remaining functions impede higher process performance. Hammer and Champy [114] and Davenport [63] therefore argue that significant improvement must always assume a holistic process perspective to be successful.

The Business Process Lifecycle

Business process management has adopted structured methods from the field of systems development as a means to effectively apply a process-oriented perspective on the operations of an organization [63]. That is, a lifecycle model has been established [289], which describes the phases that are passed in a cyclic fashion for each business process. The business process lifecycle consists of four phases that are logically interdependent as shown in Fig. 3. This conceptual model to business process management has seen wide acceptance and application, sometimes with extensions and variations, see for instance [288, 202, 91, 113, 322, 90, 80].

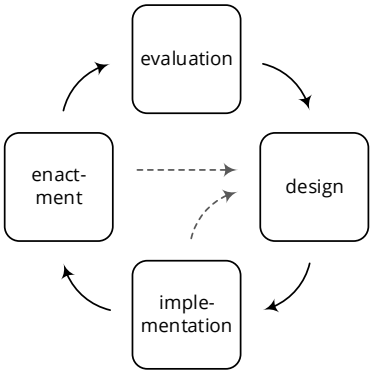


Figure 3: The Business process lifecycle.

DESIGN Upon the assumption of a process-oriented perspective, the lifecycle is typically entered in the *design* phase that strives to provide an explicit model of the business process at hand. This phase takes as input requirements from the business and envi-

ronmental constraints. As the resulting process model founds the basis for later phases of the lifecycle, it is essential that the model is correct and unambiguous. Therefore, various *analyses* are undertaken, i. e., the model is verified for its formal correctness and validated toward its precise representation of the desired process, e. g., by workshops with stakeholders and simulation.

IMPLEMENTATION Once the process model has been worked out, it can be put into action, i. e., steps are undertaken to install it in the organizational, informational, and technological environment of a company. This may range from automation using process engines to setting up operational guidelines, training process participants, and providing the required technical and organizational infrastructure to conduct the business process manually. In either case, the process model serves as a blueprint that prescribes its implementation. During the implementation phase, errors in the design of the process or issues with its implementation with a given infrastructure may be uncovered, which requires a reiteration of the design phase.

ENACTMENT After its successful implementation, the process can be run. That is, instances or cases of the process are created and handled using the infrastructure provided in the implementation phase. This is the productive phase of business processes. The state of each case can be tracked and visualized by means of the underlying process model, which enables the process-oriented perspective during runtime. During this phase, execution data of the cases is captured to evaluate the quality and performance of the process.

EVALUATION Based on the captured execution data and additional information, the business process can be evaluated. For instance, bottlenecks can be identified or the number of unsuccessful cases. Such information needs to be thoroughly evaluated and examined in order to improve the process in an iteration of the lifecycle. Evaluation may not always take place before improving or redesigning a business process. The design phase may be entered directly from enactment, if external conditions of the business change to which the process needs to be adapted.

At the center of the business process lifecycle stand the business process model and numerous artifacts related to it, e. g., documents, compliance rules, legal constraints, software components, and execution data. These artifacts need to be managed in a structured fashion in order to obtain them efficiently by various stakeholders in a business process [322].

2.2 BUSINESS PROCESS MODELS

The models created in the *design* phase of the business process lifecycle are central to business process management as they capture the knowledge to carry out actions in the context of a business process. Following [322], we define a business process model as follows.

Definition 2.3 (Business Process Model)**.**
A *business process model* specifies a set of actions and execution constraints between them, as well as their embedding in the organizational, informational, and technological environment.
A business process model acts as a blueprint for a set of cases. ◂

Business process models document business processes, that is, they capture them in a reusable form. They may be descriptive, i. e., record an already existing process, or prescriptive, i. e., specify a potential or to-be business process. A case is the instantiation of a process: a concrete execution of the described actions in the context of a business. Cases are also referred to as process instances. In this section, we give an overview of the different dimensions that have been identified for business processes and how they are accounted for in process modeling languages. Once these fundamentals of process models have been established, we discuss potential drivers and usage scenarios for business process models.

Dimensions of Business Process Models

Models are abstractions from real world phenomena and systems, reduce their complexity by neglecting irrelevant information, and map selected properties with respect to a particular purpose [269]. Business process models are typically created in order to describe the actions that lead to the desired goal, their interrelations, and the organizational, informational, and technological environment, in which the business process takes place. Consequently, these features form the dimensions of business process models commonly observed in business process management, cf. [209, 163, 322, 80].

OPERATIONS The actions and their temporal and causal relations in a business process are captured by the operational dimension, which describes the behavior of the business process. On the operational level, actions represent units of work that are performed by humans or systems and can be described informally, i. e., with a natural language label, or by precise semantic annotations [322]. Besides actions, operations also comprise events that indicate changes in the environment of the business process, such as the reception of a message, the firing of a timeout, or the occurrence of an error. In particular, start and end events demarcate the beginning and the termination of a process or

part of a process, respectively. Control flow of popular process modeling languages represents their operational perspective. As various languages differ in their expressiveness, control flow patterns have been developed that allow for their comparison in terms of behavioral features [290].

ORGANIZATION Each action of a business process is conducted under the control of some member of the organization that runs the business process, or informally, each action must be carried out by someone. Business process models capture this aspect by allocating actions and other control flow elements, e. g., decisions, to resources, i. e., individuals, or groups of individuals, such as roles or departments of a company. This allows for the utilization of resource allocation mechanisms that balance workload among process participants and find the best participant for a given task, see for instance [251].

INFORMATION Business process models are data intensive procedures. Each action has an input and produces an output that provides input to another action or contributes to the business process' result. Decisions need to be taken based on information about the process, the customer, and the process environment. This dimension captures any kind of information, such as documents but also physical material, used in the process.

TECHNOLOGY Nowadays, each business process is somehow embedded into the information technology landscape of an organization, as the majority of tasks are supported by some applications, central information systems, or service-oriented architectures. Information technology is the key enabler for effective business process management [63, 114] as it provides the glue between the actions of a process in its informational and organizational environment, i. e., technology makes information, required for a certain task, available to the responsible participant. The technological dimension of a business process is in particular relevant for its implementation, and therefore sometimes neglected in operational models.

In addition to these fundamental dimensions that are covered in almost all process modeling languages [209], more dimensions have been proposed to be accounted for in process models, cf. [131, 258, 331]. Examples are the strategic goal a process is aligned with and its risks, involved stakeholders, authorization aspects, reference processes, classification within a business ontology, estimated cost of actions, branching probabilities and execution frequency, and performance metrics, e. g., failure rate and average run time. However, these are of minor importance in the context of this thesis and are, therefore, disregarded hereafter.

Business Process Modeling Languages

Business processes can be documented in various forms, e. g., in prose text that describes in detail the actions and their execution constraints that yield a business process. Obviously, the textual description of complex business processes is inefficient as it takes a long time to comprehend the complete process and text is difficult to navigate. This is a general problem, tackled by graphical models [164]. In [226], it has been shown that graphical process models increase understanding of a process, given that the readers of these models are familiar with the modeling language. We have seen many business processes described in pictures, e. g., drawn rectangles and arrows, that can provide a better overview on the process than text, yet lack clear semantics. Often, such drawings are also not free of ambiguity, as there is no precise legend of the used notation.

Modeling therefore requires formal, i. e., precisely defined and standardized, languages that provide a limited set of constructs (notation), rules to compose them (syntax), and interpretations of correctly constructed models (semantics) [116, 202, 152]. Given a graphical process modeling language, the semantics, i. e., the behavior of a documented business process, can be derived from its graphical structure and the operational semantics of the used constructs. Academia and industry agree that standardized process modeling notations are required to properly document business processes [131, 225, 285]. Formal models are unambiguous, increase the potential for thorough analysis, and establish the basis for their enactment.

As a consequence, an abundance of process modeling languages exists to describe operational business processes that capture the dimensions operations, organization, and information [209]. The majority of these languages employs a procedural perspective on business processes that focuses on temporal relationships among actions, i. e., their execution ordering, typically referred to as control flow. Also non-procedural process modeling languages exist, e. g., case handling [292] where the execution order of actions is determined by data dependencies, declarative process models [231, 296] that implicitly specify execution order by a set of constraints, and artifact-centric process models [224, 56] that focus on central data objects and their lifecycle. However, we have not seen these approaches being widely applied, yet.

Popular modeling languages

We briefly introduce a number of procedural process modeling languages and show that they have emerged from different needs and therefore are intended to serve different purposes. For instance, Event-driven Process Chains (EPC) [142] have been developed in the context of business administration, whereas UML Activity Diagrams (UML AD) are part of the Unified Modeling Language [106] — a family of modeling languages that target at the design and engineering of information systems. The Web Services Business Process Execution

Languages (BPEL) [89] is a modeling language to design processes whose actions are implemented as web services, and explicitly aims at the model-driven execution of business processes.

The same goal is aimed at by Business Process Model and Notation (BPMN) [225], which has recently received remarkable attention as it has been developed as an integrated approach to capture operational business processes, enrich them with technical details, and enact them based on a complete specification of execution semantics. Many BPM system vendors have adopted BPMN and strive to integrate a BPMN-conforming process engine.

BPMN explained

In this work, we use BPMN to depict business processes. Figure 4 shows the order fulfillment process introduced earlier using BPMN. Actions are represented as rectangles with rounded corners and their execution order is denoted by sequence flow edges (solid arrows) that determine that an action can only start if its preceding action has been terminated. For instance, the customer's payment can only be received after the invoice has been sent to the customer. BPMN provides additional control flow constructs, i. e., events (circular nodes) and gateways (diamonds). The business process starts with a start event that denotes the reception of a message, i. e., the order. Gateways add branching semantics to control flow, where the X sign denotes an exclusive choice (or merge) between outgoing (of incoming) sequence flow edges. The parallel gateway, indicated by the + sign, activates all outgoing sequence flow edges simultaneously and synchronizes all incoming sequence flow edges. This enables the concurrent activity of product handling and finance as their tasks do not depend on each other. These concepts represent the operational dimension of business process models.

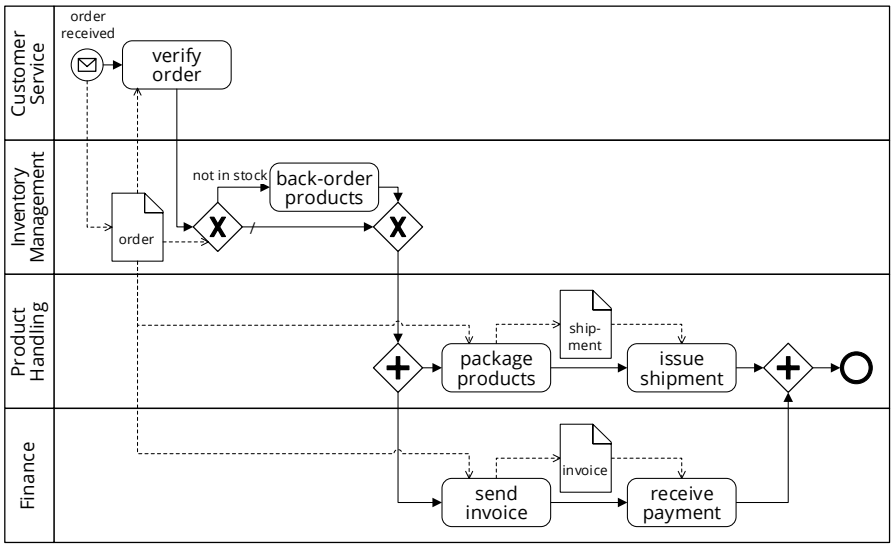


Figure 4: Order fulfillment process modeled in BPMN.

The organizational dimension is captured by horizontal rectangles, called swim lanes that represent departments of a company. A department is responsible to conduct the actions that lie within its swim lane. Finally, the information dimension is represented by the document symbols that represent data objects required for the actions of the process. Dashed arrows show input and output of actions, e. g., the action "send invoice" requires the order as input and produces an invoice as output.

Usage Scenarios for Business Process Models

Business processes are captured in process models for numerous different purposes. A recent study on the perceived benefits of business process modeling lists, among others: process improvement, understanding, communication, execution, analysis, and simulation [130]. In the remainder of this section, we will highlight some of these benefits by describing scenarios that employ business process models. Please note that this collection cannot be complete and is intended to show the spectrum of advantages provided by business process models.

COMMUNICATION OF KNOWLEDGE One of the key functions of process models is the documentation how work is performed in the context of a business process, which improves the understanding of the business process and its communication to involved stakeholders [130]. This is essential during the design of the process in order to reach agreement on a consistent view of a business process among all involved stakeholders.

Business process models serve as a reference in later phases of the business process lifecycle, cf. Fig. 3, for instance, to derive operational guidelines, or for training novices [322]. Process model abstraction, which provides consistent reduction and aggregation [235], can be used to create views on a process model, e. g., for higher management levels.

ANALYSIS Process descriptions that are grounded in a formal model show a higher potential for precise analysis [289]. In this context, verification refers to the proof of formal properties, e. g., correctness [282, 17], or compliance with business rules [102, 7]. The latter can be used for certification, i. e., the evaluation of the quality of a business by means of its processes. Tool support for verification is, for example, discussed in [87]. Validation rather focuses on the question whether the business process model meets the desires and needs of all stakeholders. Methods to validate a business process include workshops and simulation [322]. In particular to quantitatively evaluating a process model and therefore estimate the performance of the future business pro-

cess, simulation is the most popular and widely supported technique [80].

PLANNING Based on the documentation of work that needs to be done, business process models offer the capacity to plan for the resources required to run the business process profitably. For instance, if the average execution time for each action, overhead for hand-overs, the frequency of cases, and branching probabilities for choices in a process model are given, computation of the overall resource consumption in terms of working time can be computed [234], which is a precursor for planning required personnel. The same applies to other resources required, which forms the basis for costing and budgeting in the context of business processes.

Moreover, organizational changes in order to adequately support the business process may be required, e. g., the establishment of a process team that is responsible for the whole lifecycle of the process, including its correct and efficient enactment [63, 114, 91].

PROCESS IMPROVEMENT The continuous improvement of business processes is required in order to react to changes in its environment and has been reported the most important perceived benefit of business process modeling [130]. If the requirements of a business exceed the capacity of what the design of a business process is able to provide, the process needs to be improved [114]. To this end, various patterns to redesign a business process regarding to time, quality, cost, and flexibility have been presented [244]. Yet, process improvement can only be effective, if the changed process models can be put into practice immediately, i. e., without the need to develop or purchase new infrastructure. That is, processes implementation needs to be driven by the process model [266].

AUTOMATION Smith and Fingar's seminal book on the third wave of business process management praises a holistic process management on the basis of a "single definition of a business process" [266]. The "digitization" of business processes refers to model-driven automation by means of process engines that can interpret process models enriched with technical details and enact them automatically. This increases the flexibility of a company significantly: On the one hand the explicit documentation of a business process allows for the quick identification of what needs to be changed. On the other hand, its process model-driven automation allows implementing changes rapidly. Following, process improvement and process automation are complementary. This is only possible by means of information technology [289, 114, 266]. Few years ago, automation driven

by business process models has not yet been perceived as a key benefit of process models by practitioners [130], but has been stated the number one challenge of process modeling in future [131]. With the standardization of BPMN in its second version, software vendors have increasingly investigated in business process automation.

MONITORING If a business process is automatically enacted, the progress of each action can be tracked based on the execution information recorded by the process engine. At any point in time, the state of a case can be inquired and reported to the customer of the process, which increases transparency and customer satisfaction [114]. Furthermore, the continuous monitoring of business processes allows for early detection of issues, such as exceptions or unexpected delays, and quick reaction to them. In the absence of process automation, monitoring can still be provided using side effects of a process observed in information systems, such as changed documents or exchanged messages [119]. While such approaches may lack precision, i. e., in case of sparse or missing information the state of a process cannot be determined, probabilistic methods allow for the estimation of a case's state [248].

BUSINESS PROCESS INTELLIGENCE The continuous monitoring of business process instances and the integration of execution data with business process models allows for the detailed and precise evaluation of the performance of a process and the quality of its outcome. To this end, business process intelligence refers to "managing process execution quality by providing [...] analysis, prediction, monitoring, control, and optimization" [103] and combines several of the above scenarios for business process models. The information provided can be used on strategic, tactical, and operational levels, for information management and decision support [62], to steer process enactment, and drive process improvement.

Each of the addressed scenarios requires the explicit representation of the business process in a model and employs it for a particular purpose. As such, process models found the basis for effective process management in the first place and are the key enabler for the thorough analysis, automation, monitoring, and improvement of the operations of a business. Since the act of modeling business processes is a demanding and time-consuming task, these models shall be stored and made accessible for reuse.

2.3 PROCESS MODEL REPOSITORIES

Business process models constitute significant value for an organization as they are the central knowledge asset of business process management. In particular, large companies with hundreds or even thousands of business process models [250, 3, 77] require effective and efficient means to manage these models. Process model repositories have been proposed to store these models and make them available to their stakeholders [108], and are part of business process management systems [322, 80]. We illustrate their role for business process management in Fig. 5 that depicts the generic architecture of a business process management system (BPMS), based on [322].

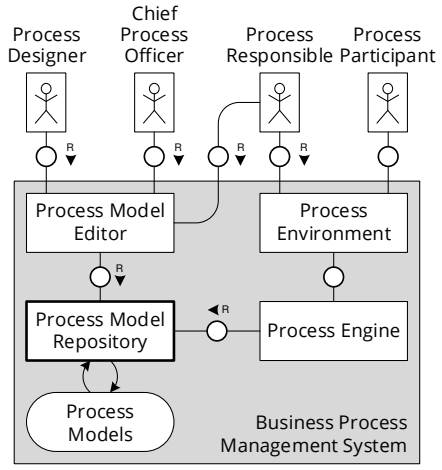


Figure 5: Conceptual architecture[1] of a business process management system, where the process model repository comprises the central component.

Stakeholders of business processes

A number of stakeholders are involved in business process management [322], of which some are discussed here. The process designer is an expert in process modeling and documents the business process with the input of knowledge experts, using a process model editor. The process model editor is a software application to access process models, i. e., provides the user interface to search, retrieve, and present process models, facilitates their maintenance and distribution, and allows for their manipulation and analysis [18]. Once a process model has been implemented, it can be accessed by a process engine and enacted, cf. Sect. 2.1. Process participants conduct the actual operations of a process and may not be aware of the complete process, as they are in charge of parts thereof, only. They interact with the process engine through the technical process environment, e. g., worklist applications, document management systems, etc. Following the concepts of resource allocation [251], the environment may even order the participants to conduct certain tasks.

1 Figure 5 is modeled using FMC [147]; see Appendix A for a brief explanation.

The process responsible — a tactical position in an organization — is responsible to control the lifecycle of a particular business process including the phases design, implementation, enactment, evaluation, and process improvement. The process responsible therefore needs access to both the process model and runtime information. The chief process officer works in strategic management and is responsible for the evolution, harmonization, and standardization the process landscape of an organization.

Repositories

In the center of the applications of business process models stands the process model repository that stores and manages them along with related artifacts that originate from the dimensions of business process models. Process model management is seen as one of the major challenges [131] of business process management. In their pioneering work, Bernstein and Dayal define a repository as "a shared database of information about engineered artifacts produced or used by an enterprise" that provides specific access and management functions beyond the features of a database system [31]. Some basic functionality, e. g., storing, updating, retrieval, and deletion, access management, version and transaction management are common for repositories. Moreover, effective business process management requires additional features [258, 331], e. g., support for the business process lifecycle, cf. Sect. 2.1, a canonical process model format that unifies various process modeling languages [163], and the management of artifacts that are related to the process model but not part of it [77]. A comprehensive survey of academic and industrial process model repositories has been conducted in [331], from which the authors deduce a holistic repository management model. This management model fueled the design of a reference implementation for process model repositories [163].

Repository use cases

In the remainder of this section, we outline some use cases of business process model repositories that are illustrated with their respective stakeholders in Fig. 6 using UML Use Case diagrams [106].

As mentioned before, the process responsible is in control of the business process lifecycle. Besides being in charge of the correct and efficient enactment of business processes, the process responsible steers the process model's design, implementation, evaluation, and re-design. Therefore, they need access to the process models. This is captured in a collective fashion by the use case *manage process model* and comprises functionality for the creation, update, deletion, retrieval, import, and export of models. Process participants may also desire to *get insight* into process models to get an understanding of the context of their work. Model retrieval requires some form of *navigation*, e. g., a directory that lists stored process models by some classification or means to search for process models.

In particular the design of new business process models benefits from harnessing existing knowledge in the process model repository.

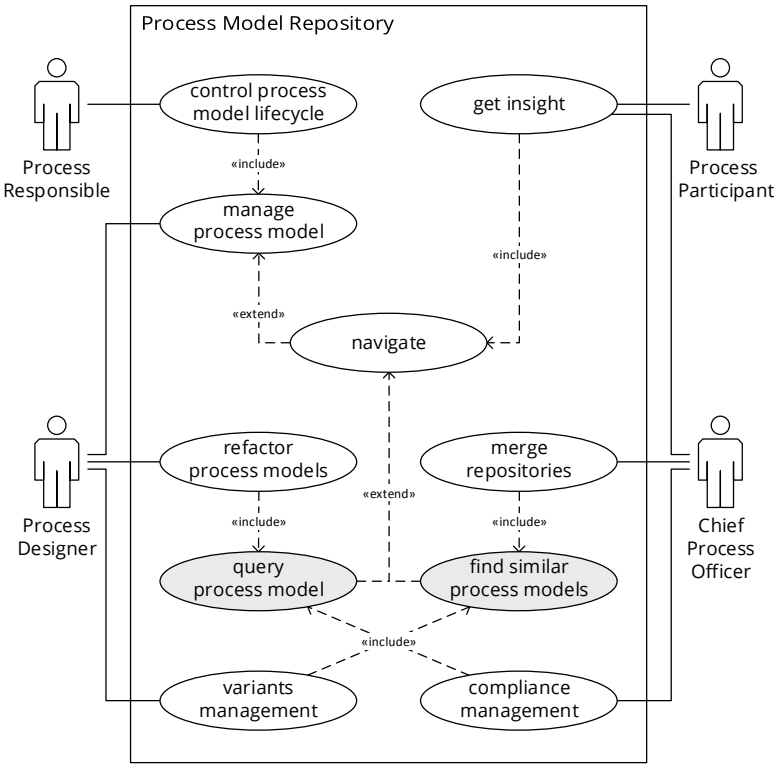


Figure 6: Use cases of a business process model repository.

That is, if a model to be newly created shares aspects with existing models in the repository, these synergies should be used, as they can improve the quality of process models [100], and reduce time and errors [3, 240]. Tool support is provided, for instance, by the automatic completion [149, 192] or substitution of process model fragments [192].

Iterative improvement of process models tends to introduce redundancies among process models and increase their complexity, which makes further changes increasingly difficult and prone to anomalies. These issues can be tackled by *refactoring process models* [311], which includes, but is not limited to, the unification of certain procedures among process models, the extraction of fragments common to several process models into a subprocess, the removal of unused process model fragments, and the propagation of changes [161, 311]. In order to apply refactoring operations consistently, features to discover all process models that show certain structural or behavioral aspects are required. We refer to this as *querying process models*.

A second use case that concerns the process designer is the *management of process variants*. It is unlikely that a single process definition fits all requirements of a business process [114] and therefore several versions or customizations of a single process exist. For instance, order fulfillment may be carried out differently for customers that have a long standing relationship with a business, compared to new

customers that only order small quantities of goods. Process versions may be treated individually and related to a reference model [88, 83] or they may be incorporated into a single, configurable model, where a configuration denotes a particular variant [294, 162]. Discovering variants and deriving a reference or merging them into a configurable model requires means to *find process models that are similar* to each other [204, 221, 249].

The chief process officer, as a manager of all processes and process models of a business, also needs insight into the process models. Mergers with or acquisitions of other companies require consolidating their process repositories into a consistent process model collection, i. e., *identify similar models* and choose one of suitable candidates or merge several models into one. *Compliance management*, i. e., verifying the adherence of business processes to legal or business rules [102, 7], again requires means to *query process models*, whereas the query expresses particular rules that need to be checked against the collection of process models.

From the above discussion, we observe that process model search is an essential feature of process model repositories, as virtually any use case requires the retrieval of relevant process models from a potentially very large collection.

2.4 SEARCHING BUSINESS PROCESS MODELS

Search is defined as the act of "finding something by looking or otherwise seeking carefully or thoroughly"[2]. Following this definition, process models of a particular interest can be searched manually by carefully looking at each model stored in the repository for relevant aspects. With modern companies maintaining hundreds or thousands of business process models [250, 3, 77], this is infeasible, and tools that carry out this task are required. With regard to this, we refer to the search problem from an algorithmic perspective, i. e., assume a software system that takes a search question as input, retrieves responses to this question, and returns them to the user in an automatic fashion.

Given a specification of the search question — the query — search addresses the retrieval of business processes that best match the given specification and rank them such that relevant models are presented first [336, 194]. To this end, search can be successful or unsuccessful. If search is successful, it must return all process models that are relevant for the query. Unsuccessful search does not denote that relevant models were not found, but rather that no models relevant to the query exist [148].

However, search does not only address the process of finding answers to a search question, but also the collection and management

2 according to the *Oxford Dictionary of English*

of information that enable the retrieval of these answers [148]. This includes the organization of information provided about a business process in such a way that their fast retrieval is possible and also reflects the latest version of process models stored in the repository. Not all information provided about a business process model is relevant for search and may even inhibit efficient search. Hence, features relevant for process model search shall be extracted, whereas remaining information can be neglected. For search to be effective, the extraction of relevant features must not put additional burden on process model designers, whose work shall be made available for search.

If all these aspects are met, process model search provides considerable value to the reuse and repurposing of the stored process model knowledge, for instance, reducing redundancies among various process models and increasing the quality of newly designed models, which have been shown in a user study [150]. This section presents the background to process model search in general and prepares the ground for searching process models by example.

Dimensions of Search

Process models can be searched for by any property they reveal, including the aforementioned dimensions operation, organization, information, and technology. For instance, a query may request all business processes, where the action "verify order" is carried out in the customer service department. Such information can be extracted from process descriptions in a straightforward manner. Moreover, the majority of such queries could be answered with search features of general purpose databases. For instance, properties of certain type are stored in a relational database and searched for with a structured query language.

However, such simple approaches are incapable of answering queries with respect to the semantics of a business process. The term "semantics" frequently causes confusion [116] and is used in the literature for approaches that leverage ontologies to identify concepts, e. g., [81], approaches that use dictionaries to account for terminological heterogeneity, e. g., [94, 128], approaches that consider the behavior of a process, e. g., [135], or approaches that just use a technology from the field of semantic web, e. g., [129, 265]. Therefore, we refrain from using the term "semantic" in this context and rather focus on the concepts expressed through the used *text* in action labels, the process' graph *structure*, and the *behavior* described by the process model [79].

Free-text & classification

Among the most simplistic methods to collect information and retrieve answers for a query is free-text search [117]. That is, a parser extracts terms from a document, e. g., the labels of actions in a business process model, and manages a list thereof that can be quickly searched and points to models containing these terms. However, such methods rely on regularities in linguistics and require long, con-

tinuous texts to be statistically accurate [117]. Both conditions are met only to a limited extent by process model collections to which many individuals contribute. Issues of terminological heterogeneity can be tackled by natural language processing [187] to overcome, for instance, synonym relations, cf. [81, 300, 167].

A way to facilitate navigation, frequently seen in practice, is the classification with regard to functional aspects or business domains, e. g., logistics, operations, marketing and sales, and customer service. Approaches to classify information, e. g., business processes, assume that there exists a uniform way, a universal answer on how classes can be derived from given information and which classes to assign to which information. However, no classification can correctly represent a constantly evolving information space [117] and it is virtually impossible to allocate unique classifications that are consistently agreed upon by all users of such a system [92].

Structure

The majority of process modeling languages comprises a graphical notation [209], i. e., process models are comprised of nodes and edges of various types that have a particular sense. This allows using approaches to graph matching [40, 207] to evaluate how much two process models resemble each other structurally and to decide a match, evaluating whether actions contained in both models are similarly connected.

Searching and comparing process models by their graph structures has been proposed manifold in the literature on process model search, cf. [22], and has been shown to provide good results in terms of providing relevant search results [105, 75, 82]. Structural methods are, for instance, useful to searching a repository using structural patterns [109, 328] or to obtain fragments to complete a process model under design [81].

Behavior

Comparing the structure of a pair of process models may be misleading, as two models that appear to be similar in their structure may describe different processes [66] and models that are structurally different can, nevertheless, describe identical behavior [21]. Therefore, researchers have also turned their attention to behavioral approaches for process model search. The behavior of a process is expressed by the execution conditions of the constructs used in the process model graph, and can be captured by sequences of observable side-effects or exchanged messages, e. g., [141, 170], the space of reachable states, e. g., [267, 221], or abstractions thereof, e. g., [298, 314].

We explore various concepts to capture and compare the behavior of process models in Sect. 4.2, and discuss approaches to process model search concerning the above dimensions in Sect. 5.2 and Sect. 6.2 in detail.

Methods for Search

We observe two search methods in the literature [77], *similarity search* and *querying*, that differ essentially in the way a query is compared with a candidate. These two search methods have already been introduced as basic functionality of process model repositories, cf. Fig. 6.

Similarity search

In process model similarity search, the query is a regular process model that describes a complete business process and comparing it with a candidate process model from the repository yields a measure that quantifies how much these two models resemble each other. Similarity search returns a set of complete models that are not required to match a query exactly, and the result set is typically ranked by their similarity to the query. Therefore, it is also referred to as approximate search and is, in particular, useful to detect duplicates or very similar process models in a large collection.

While related work rather focuses on meaningful similarity measures for process models, the efficient retrieval during search remains a challenge, due to the complexity to compare the structure or behavior of a process model.

Querying

Querying, in contrast, supports cases where few yet essential aspects of a search question must be met precisely by a match, i. e., the query is not required to be a complete process model. If, for instance, a query is superseded by a process model that has significantly more features, the model will not be considered as a similar model, although the query is matched precisely.

A query can be expressed in any structured form, e. g., using a textual or visual query language [6], or with a regular process model itself that only comprises desired parts of a process [154]. Query results may be complete models, i. e., all models that meet the requirements of a query, or fragments thereof, i. e., only the portion of a model that matches the query. The latter is not trivial, as it requires a projection of the search question to the process model graph.

The Business Process Lifecycle Revisited

After a characterization of the major search dimensions, i. e., what can be searched for, and prevailing search methods, i. e., which type of search question can be asked, we now turn to the management of process model search, i. e., the collection of required information and their allocation that enables efficient search.

In Sect. 2.1, we have introduced the lifecycle for managing business processes, comprised of the phases design, implementation, enactment, and evaluation in an iterative fashion. Following, we have argued that business process models, created in the design phase, are essential for virtually all activities in the context of business process management, cf. Sect. 2.2, and are maintained in process model repositories, cf. Sect. 2.3.

Organizing process models for search

Enabling search among process models stored in a process repository requires a representation of all characteristics that are relevant for a search question in a machine-interpretable representation [118]. To this end, we assume process models captured in a process modeling language with formal semantics and stored in an electronic process model repository, and hence being already in a machine-interpretable form. However, their organization is not optimized for search, i. e., in order to decide, whether a model matches a given search question, each model needed to be examined exhaustively with regard to that question. Instead, an *index* provides the organization of process models targeted at search: For a particular search dimension, e. g., classification, structure, or behavior, the index maintains references to process models along with characteristic features to answer queries without exhaustively examining each model.

Management of process model search is also organized in a lifecycle illustrated in Fig. 7, cf. [36, 191], that is closely coupled with the business process lifecycle introduced above.

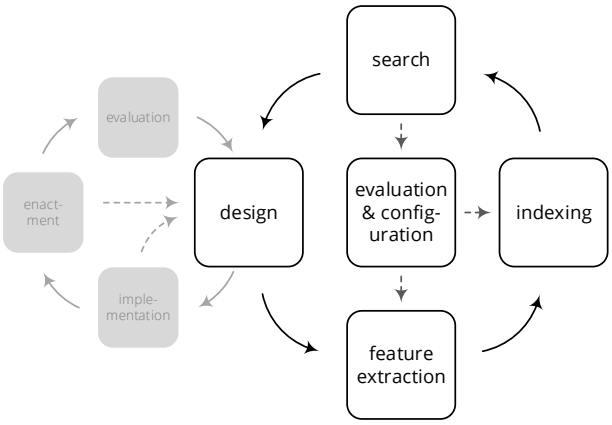


Figure 7: The process model search lifecycle in the context of business process management.

SEARCH Searching the process model repository for already existing solutions to a problem or parts thereof should be conducted as a precursor of the design or re-design of a business process model. Below, we will explain the stages of a search run in greater detail.

DESIGN The lifecycle of a business process meets the lifecycle to manage the according model for search in the design phase of the process model. Each time a model is manipulated, i. e., created, changed, or deleted, the index must be updated to reflect the latest version, or several versions, of a process model in future searches.

FEATURE EXTRACTION Once the manipulated process model has been submitted to the repository, it should be analyzed for fea-

tures that are relevant for search. Such features can be provided by humans, e. g., a classification within a business domain or the annotation of a process model with certain keywords [117]. Classification and annotation often need to be conducted manually and are, therefore, costly and subject to the skills and preferences of the individual who adds this information [92]. As a consequence, this may lead to an inconsistent characterization of the models within the repository. Instead, the features should be extracted automatically. For classification or text-based search, this can be done, for instance, by natural language processing of the used vocabulary [187]. Graphical features and the behavior can be obtained from the process models and be leveraged for search [79, 22, 308].

INDEXING The extracted features are then stored in an index, such that, during search, candidates can be retrieved in an efficient manner, i. e., without comparing the search question with each process model in the repository [276]. As we show in this thesis, the actual type of index depends primarily on the data that is searched for and the type of search, i. e., querying or similarity search. For instance, indexes for similarity search can cope with data that has no inherent ordering, which would make the application of traditional indexes, e. g., binary trees, infeasible [333].

CONFIGURATION AND EVALUATION The above phases are involved in the management of process models for search in productive process model repositories. Yet, searching for process models should be effective and efficient. This requires an evaluation of the search approach with respect to its quality, i. e., the relevance and quality of search results, and its performance, i. e., resources consumed during search and its scalability for large numbers of process models in the repository. These criteria can be evaluated by experiments involving search and their results can be used to improve the quality by configuring feature extraction and matching, and performance by configuring the index.

Note that the search lifecycle is not iterated entirely for each process model, i. e., after a model's design and extraction of its features, it will be updated in the index. Future search then takes the complete index, i. e., the representation of all models in the repository, as an input for another iteration of the lifecycle. Searching process models by various dimensions may require several indexes, depending on the search method and the process models' dimensions of interest, as argued above.

Search workflow

The search phase itself takes the user's search question, i. e., a query model, as input and is comprised of several steps to retrieve process models from the repository that match the given query model, depicted in Fig. 8. Related work on process model search proposes var-

ious types of query models, e. g., keywords, structured text queries, graphical query models, and regular process models. We discuss this related work in Sect. 5.2 and Sect. 6.2.

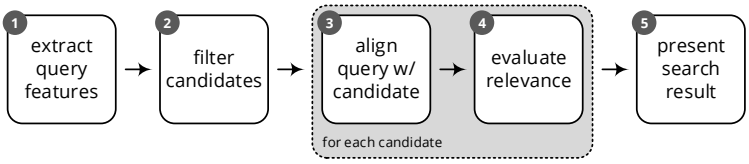


Figure 8: Steps of the search workflow, comprised in the *search* phase of Fig. 7.

(1) The first step of search consists of processing the input query, i. e., extracting the features that compose the search question. Based on these features and the index for process model search, (2) a set of candidate models that may be relevant to the query are obtained. Here, the index allows for efficient search if irrelevant models can be excluded from the remaining search process.

For each candidate, (3) an alignment with the query must be constructed, i. e., the features requested in the query must be mapped to corresponding features in the candidate process models. Process alignments and their construction are a research topic of its own and not in the focus of this thesis. We explore process alignments briefly in Sect. 3.4. (4) Based on the alignment, the relevance of the candidate with regard to the query can be evaluated and irrelevant candidates excluded.

As a result of steps 3 and 4, a set of relevant process models is returned as the search result for the query. (5) In the final step, these models are ranked by their relevance to the query and presented to the user. This includes processing the matches to support the user's comprehension of the result proposed for their search question.

In this thesis, we focus on the behavior of process models in terms of the execution ordering of actions. Chapter 4 introduces the concepts of behavioral distances to search for process models, and explains, how corresponding features are extracted from process models (1). This founds the basis for the efficient retrieval of candidates (2) and deciding a match (4). For the latter, we present solutions concerning querying process models by example in Chap. 5 and process model similarity search in Chap. 6. Section 4.4 and Sect. 4.5 present solutions to assess the quality of a search result with regard to its ranking, and to visualize the matched aspects of the query in models of the search result. An evaluation of the proposed solutions is provided in Chap. 7.

Challenges & Requirements of Process Model Search

To conclude this section, we elaborate on prevailing challenges and requirements toward process model search. Comparing process models

is far from trivial, as it requires the construction of an alignment that connects features of the query with features of a candidate model. Construction of an alignment is closely related with the matching problem [86] and has been studied in the area of ontology matching [242, 262].

Challenges of heterogeneity

Aligning process models is challenged by different types of heterogeneity [86]: syntactic, terminological, and conceptual. Syntactic heterogeneity refers to the usage of different modeling languages to express a process model, e. g., BPMN, EPC, and UML AD. While each of these languages has particular features, they share a large ratio of common concepts. Differences can be mitigated by the use of a canonical process model format [163] or by Petri net based formalisms [177]. Terminology refers to the vocabulary used to label process model elements, such as actions. Following the argumentation of [92] the absence of central control of this vocabulary leads to a considerable degree of heterogeneity in the labeling of process models. A large body of work exists that strives to resolve these issues. We abstract from this problem and provide a brief discussion in Sect. 3.4.

Conceptual heterogeneity denotes differences in perspective and granularity of business process models. Process models may be designed for a certain purpose and may only represent a certain perspective, e. g., the interaction with the customer instead of internal actions or the alignment of process actions with information systems. Granularity refers to the level of abstraction applied in business process models. A solution proposed for this issue is the concept of complex correspondences, i. e., aligning an aggregated set of features of one model with a single feature or an aggregated set of features of another model [316]. While differences in the granularity of two models can be handled by automatic approaches to a limited degree, their implications to process model search are not straightforward. Therefore we resort to comparing process models that show a similar degree of abstraction.

Requirements for search techniques

Beside these challenges, authors of various process model search approaches have demanded a set of requirements that shall be satisfied to make the search approach useful in practice. We briefly discuss them in the following and refer to them throughout this thesis.

INTUITIVE REQUEST SPECIFICATION Formulating the query needs to be intuitive and the search interface easy to use [192, 25], which is closely related with the capabilities of the intended audience of a search approach. Search can only return results of high quality, if the users can precisely express their search question [146]. For instance, process modeling experts are likely capable of comprehending and applying complex and expressive query languages, whereas business analysts, process owners, and process participants may not be able to express themselves in such a way.

COMPREHENSIVE DISCOVERY MODEL The discovery of results, i. e., how a search question is resolved against the process models stored in the repository, needs to be comprehensible and tractable by the user, in order to rely on the results that are provided [100, 194] and to refine queries that did not provide a desired result [192]. While this requirement concerns the mechanisms to evaluate relevance of a candidate and decide a match, it is closely coupled with the request specification. That is, assumptions made by the discovery model that users are not aware of, as they are not explicitly reflected in the query, are likely to confuse.

DIVERSITY As any possible search question cannot be anticipated, process model search should span a diversity of search dimensions and search methods [84, 26]. However, comparing structure or behavior of business process models is not trivial and requires particular methods and techniques. For instance, process model similarity search and querying introduced in this thesis, require completely different indexes. A common search platform that integrates various search engines, can provide a uniform interface and thereby provide the deliver diverse usage opportunities.

ABSTRACTION Abstraction generally refers to the reduction of information on relevant aspects. The same holds for process model search [84]. Aspects that are not relevant for answering a query, e. g., the particular process modeling language, should be ignored by the system.

RESULT RANKING Mentioned before, the ranking of matches is essential to providing a search result of high quality, which includes expressing the degree how well a given process model matches the query [100, 192, 181]. For similarity search, rankings can be provided implicitly, ordering matches by their similarity to the query in descending order. Querying, however, requires an explicit ranking, as the majority of approaches in the literature only provides a binary answer [97].

FACILITATE REUSE Once a search result set has been obtained and ranked, it needs to be presented in a way that supports comprehension by the user and facilitates reuse of the proposed business process models or fragments. On the one hand, this addresses highlighting those parts of a process model that contributed to the match, which enables the user to assess the relevance of the matches [156]. On the other hand, the automatic extraction and incorporation of these parts into a process model currently under design contribute to their reuse [25].

SCALABILITY Finally, search must be fast and scale well with large process model collections [192, 84, 99]. Efficient search can only be achieved by using indexes that avoid exhaustive examination of all models in the process model repository.

3

FOUNDATIONS OF PROCESS MODEL SEARCH

Before we can present our approach that addresses the requirements raised in the previous section, we need to introduce a number of formal concepts our work builds upon.

After a clarification of notations for basic mathematical concepts in Sect. 3.1, we introduce Petri net systems as a means to capture and express the semantics of dynamic systems in a powerful and well-established formalism in Sect. 3.2. Based on the execution semantics of net systems, we elaborate on successor relations that have been proposed as an abstraction of the behavior of business processes, cf. Sect. 3.3. We rely on these relations, as they found the basis to identifying and quantifying commonalities among business processes.

Section 3.4 introduces process alignments that capture correspondences between a pair of process models. Only when such correspondences have been established can business process models be compared, and therefore they are fundamental to our work. Since work on process alignments is a research area on its own, we also report on its background and recent contributions toward their automatic construction.

3.1 SETS, RELATIONS, AND SEQUENCES

In this section, we briefly recall basic mathematical concepts and their notations that are used throughout this thesis.

Notation 3.1 (Sets). A *set* $S = \{s_1, ..., s_n\}$ is a collection of distinct objects. We refer to the cardinality of a set by $|S|$, and the empty set by $\emptyset$. A set is finite if its cardinality is bounded. The powerset of S, i. e., the set of all subsets including $\emptyset$, is denoted by $\mathcal{P}(S)$. For two sets $=$ denotes their equivalence, $\subseteq$ their inclusion, $\cap$ their intersection, $\cup$ their union, and $\times$ the Cartesian product over both sets. The set difference is denoted by $\setminus$. ◂

We refer to the set of natural numbers including 0 by $\mathbb{N}$, and the set of real numbers by $\mathbb{R}$. A closed interval of these sets is denoted by $[a, b]$, where $a, b \in \mathbb{N}$ or $a, b \in \mathbb{R}$, respectively. For a value $x \in [a, b]$ holds that $a \leq x \leq b$. The open interval (a, b) is defined analogous, whereas $x \in (a, b)$ denotes that $a < x < b$. Open and closed intervals can be combined, e. g., $x \in (a, b]$ implies $a < x \leq b$.

Boolean algebra operations use the following notation: $\wedge$ denotes the conjunction of boolean statements and $\vee$ their disjunction. Implication is denoted by $\implies$ and the equivalence of statements is expressed by $\iff$.

Notation 3.2 (Relation). An n-ary *relation* is defined as $R \subseteq S_1 \times ... \times S_n$, $n \in \mathbb{N}$, $n \geq 1$, and denotes a set of n-tuples, such that each element consists of n components and the i^{th} component, $i \in \mathbb{N}$, $i \in [1, n]$, is an element of S_i. For *binary relations* holds $n = 2$. ◂

A binary relation $R \subseteq S \times S$ can assume the following properties.

- R is symmetric iff $(x, y) \in R \iff (y, x) \in R$
- R is reflexive iff $\forall\, x \in S : (x, x) \in R$
- R is irreflexive iff $\forall\, x \in S : (x, x) \notin R$
- R is antisymmetic iff $(x, y) \in R \implies (y, x) \notin R \vee x = y$
- R is transitive iff $(x, y) \in R \wedge (y, z) \in R \implies (x, z) \in R$

R^+ denotes the irreflexive, transitive closure of R.

A binary relation $f \subseteq (S_1 \times S_2)$ is a *function* that maps elements of S_1 to elements of S_2, denoted by $f : S_1 \mapsto S_2$, iff $\forall\, x \in S_1\ \exists\, y \in S_2 : (x, y) \in f$ holds and $\forall\, x \in S_1,\ y_1, y_2 \in S_2 : (x, y_1) \in f \wedge (x, y_2) \in f \implies y_1 = y_2$ holds. We write $f(x) = y$ if $(x, y) \in f$.

A binary relation $R \subseteq S_1 \times S_2$ can be expressed as a *matrix* of $|S_1|$ rows and $|S_2|$ columns, which are arbitrarily ordered, such that, the i^{th} element s_i of S_1 is represented by the i^{th} row of the matrix, and the j^{th} element s_j of S_2 is represented by the j^{th} column. An element $(x, y) \in R$ is located at the intersection of the i^{th} row and j^{th} column,

if $x = s_i$ and $y = s_j$. A value in the matrix is undefined, illustrated by $\cdot$, if $(s_i, s_j) \notin R$.

Notation 3.3 (Sequence). A *sequence* is a function $\sigma : \{1, ..., n\} \mapsto S$, $n \in \mathbb{N}$, over a finite set S, also referred to as $\sigma = \langle s_1, ..., s_n \rangle$. The length of sequence σ is denoted by $|\sigma| = n$. S^* denotes the set of all finite sequences over S, including the empty sequence $\langle\rangle$. ◂

We write $x \in \sigma$ if $\exists\, i \in \mathbb{N} : \sigma(i) = x$. The *concatenation* of a sequence $\sigma \in S^*$ and an element $s \in S$ yields a new sequence σ', denoted by $\sigma' = \sigma; s$, and is defined as $\sigma' : \{1, ..., |\sigma| + 1\} \mapsto S$, such that for $1 \leq i \leq |\sigma| : \sigma'(i) = \sigma(i)$ and $\sigma'(i+1) = s$.

3.2 NETS AND NET SYSTEMS

Petri nets are graphs that convey the behavior of dynamic systems. Carl Adam Petri formulated the foundations for Petri nets in his seminal PhD thesis [232] in 1962 as a theory for the communication of systems, thereby extending automata concepts by the notion of concurrency. Petri nets comprise a simple graphical notation and a strong mathematical foundation. A large body of literature is dedicated to Petri nets, e. g., [112, 246, 218, 68], that have shown a number of important properties.

Syntax and Semantics of Petri Nets

Process models may be captured as workflow systems [283], a dedicated class of Petri nets. For many common process description languages, e. g., BPMN, EPC, and BPEL, net-based formalizations have been presented, see [177] for an overview. Based on [283], we recall basic notions of nets and net systems.

Net syntax

A Petri net, or net, is a directed, bipartite graph comprising nodes that are distinguished into places and transitions and directed edges connecting them.

Definition 3.1 (Petri Net)**.**
A *Petri net* is a tuple $N = (P, T, F)$ comprised of disjoint, finite sets of places P and transitions T, and a flow relation $F = (P \times T) \cup (T \times P)$. ◂

For a node $x \in (P \cup T)$, $\bullet x = \{y \in (P \cup T) | (y, x) \in F\}$ denotes its *preset* and $x\bullet = \{y \in (P \cup T) | (x, y) \in F\}$ its *postset*. Consequently, $\bullet t$ comprises the set of input places for transition t and $t\bullet$ its set of output places.

F^+ is the irreflexive, transitive closure of the flow relation. A *path* $(x_1, x_n) \in F^+$ is a sequence $\langle x, ..., x_n \rangle$ with $\forall\, 1 \leq i < n : (x_i, x_{i+1}) \in F$. A path $\langle x_1, ..., x_n \rangle$ is referred to as cycle or circuit, if $(x_n, x_1) \in F$. A net is strongly connected, if $\forall\, x, y \in (P \cup T) : (x, y) \in F^+$.

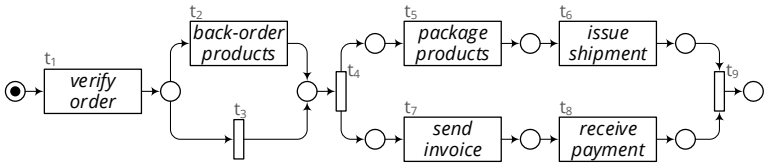


Figure 9: Example net system for the order fulfillment process.

An example Petri net, illustrated in Fig. 9, represents the order fulfillment process introduced in Sect. 2.1. Places of the net are depicted by circles that may contain tokens, visualized as black dots. For now, we ignore tokens and return to them later. Places are connected to transitions, denoted by rectangles, in a bipartite fashion, i. e., places are only connected with transitions and vice versa. Actions and events of a business process model are represented by transitions in a Petri net. Hence, we labeled a number of transitions with the respective task. Some transitions have no label, as they do not represent an action, but are rather required to preserve the behavior of the business process. We resort to empty labels for the sake of visualization, but nevertheless assume each transition to be unique in our formal model, denoted by the identifiers t_1 to t_9 in Fig. 9.

Net semantics

Based on the syntax of a net, we define its semantics in terms of markings. The term *marking* refers to the assignment of tokens to places and provides for the behavioral semantics of Petri nets. Tokens are visualized as black dots located on places in the graphical representation of net systems and represent execution conditions for transitions in Petri nets. If all input places of a transition carry a token, the transition can be executed, by which it consumes a token from each input place and produces one on each output place.

Definition 3.2 (Marking, Net System)**.**
Let $N = (P, T, F)$ be a Petri net. A *marking* of N is a function $M : P \mapsto \mathbb{N}$, such that $M(p)$ returns the number of tokens in place p. $\mathbb{M}$ is the set of all markings of N.
A *net system* is a pair $S = (N, M_0)$, where N is a net and M_0 is the initial marking. ◂

Let $M, M' \in \mathbb{M}$. We write $M \geq M'$ if $\forall\, p \in P\colon M(p) \geq M'(p)$. $M + M'$ denotes a marking M'' such that $\forall\, p \in P : M''(p) = M(p) + M'(p)$. We also use the term "system" for net systems, if its meaning is obvious from the context.

Definition 3.3 (Enabling, Firing)**.**
Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$ and $M \in \mathbb{M}$.
A transition $t \in T$ is *enabled* in M, denoted by $(N, M)[t\rangle$, iff $\forall\ p \in {\bullet}t : M(p) \geq 1$.
A marking $M' \in \mathbb{M}$ is reachable from M by *firing* an enabled transition t, denoted by $(N, M)[t\rangle(N, M')$, such that $\forall\, p \in {\bullet}t : M'(p) = M(p) - 1$ and $\forall\, p \in t{\bullet} : M'(p) = M(p) + 1$. ◂

A sequence of transitions $\sigma = \langle t_1, ..., t_n \rangle$, $n \in \mathbb{N}$ is a *firing sequence*, iff $\exists\, M_1, ..., M_{n+1} \in \mathbb{M}\ \forall\, 1 \leq i \leq n : (N, M_i)[t_i\rangle(N, M_{i+1})$.

For any two markings $M, M' \in \mathbb{M}$, M' is reachable from M, denoted by $M' \in [N, M\rangle$, if there exists a firing sequence σ that is enabled in M, denoted by $(N, M)[\sigma\rangle$, and leads to M'. Then, firing of σ is denoted by $(N, M)[\sigma\rangle(N, M')$.

Markings express execution conditions for transitions, and firing of transitions leads to a new marking, i. e., a marking represents the state of a net system. Hence, the set of markings reachable from the initial marking M_0 represents the state space of a net system. To find out, whether a transition can be executed, a reachable marking that enables this transition needs to be found. In general, this requires constructing the state space of a net system and examining whether such a marking can be reached [112].

Although Petri nets have been devised for concurrency, we assume sequential firing semantics, referred to as trace semantics [125]. In Petri nets, firing a transition is transactional and only one transition can fire at a time, which provides us with the notion of firing sequences. Concurrency in a net is expressed by several transitions being enabled in a marking, of which one is chosen to fire. Subsequently another of the enabled transitions can fire. The interleaved firing of transitions leads to an exponential growth of the number of reachable markings [280].

Based on firing semantics, we can characterize the behavior of a net system by the set of traces it can produce [125], i. e., all distinct firing sequences that are enabled in the initial marking.

Definition 3.4 (Traces)**.**
Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$ and $M_0 \in \mathbb{M}$. The set of *traces* of a system is defined by $\mathcal{T}(N, M_0) = \{\sigma \in T^* | \exists\, M \in \mathbb{M} : (N, M_0)[\sigma\rangle(N, M)\}$. ◂

In our definition, we do not require that a trace describes complete firing sequences, i. e., that the last transition occurrence in a trace represents termination of the net system, for the sake of simplicity. Example traces of the system depicted in Fig. 9 are $\langle t_1, t_2, t_4, t_5, t_7, t_8, t_6, t_9 \rangle$ and $\langle t_1, t_3, t_4, t_5, t_6, t_7, t_8, t_9 \rangle$. Here, transitions t_5 and t_6 can be concurrently enabled with transitions t_7 and t_8, which yields a number of different orderings in the set of traces.

Properties of Nets and Net Systems

A number of concepts are based on particular classes of net systems and their properties, which have been proven elsewhere. In the following, we briefly repeat their definitions for reference.

Workflow nets

Workflow nets have been introduced for the verification of process models [282] and are a specific class of Petri nets that ensure precise instantiation and termination semantics, as they restrict the net to a single initial and final place, respectively, and require a degree of structural correctness.

Definition 3.5 (Workflow Net)**.**
A net $N = (P, T, F)$ is a *workflow net*, iff N has a start place $i \in P$ such that $\bullet i = \emptyset$, a final place $o \in P$ such that $o\bullet = \emptyset$, and the short-circuit net $N' = (P, T \cup \{t_c\}, F \cup \{(o, t_c), (t_c, i)\})$ is strongly connected. ◂

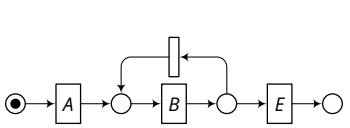
A system $S = (N, M_0)$ is called a workflow system if N is a workflow net.

Example workflow nets are depicted in Fig. 10. Each workflow net comprises exactly one initial place, one final place, and every node lies on a path from the initial to the final place.

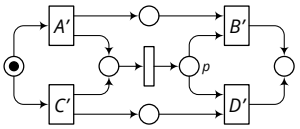
Free-choice

Another restriction on the structure of a net is the *free-choice* property [68], which suggests a compromise between the expressive power and the analyzability of Petri nets [282]. As free-choice nets show a close coupling of structure and behavior [145], they allow for more efficient analysis. The term free-choice refers to the requirement that every choice can be made freely from the perspective of involved transitions, i. e., all transitions that are involved in a choice have the same knowledge about their enabling. If several enabled transitions share a common input place, they compete for it, since the firing of one of these transitions consumes the token, and therefore, disables the other transitions.

This is visualized in Fig. 10b. If a token is in place p, transitions B′ and D′ compete for it. However, the choice, which of these transitions can fire, cannot be made freely. For instance, B′ has another input place that is only marked, if A′ has fired before. The same holds for D′ and C′. In fact, the choice, whether B′ or D′ can fire, is already taken in the first choice between A′ and C′.



(a) A sound, free-choice workflow system.



(b) A sound workflow system that is not free-choice.

Figure 10: Example workflow systems.

Definition 3.6 (Free-Choice)**.**
A net $N = (P, T, F)$ is *free-choice*, iff $\forall\, t_1, t_2 \in T : {}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset \implies {}^\bullet t_1 = {}^\bullet t_2$. ◂

A net system $S = (N, M_0)$ is free-choice, if N is free-choice.

It is clear from the definition that, whenever an enabled transition that shares at least one input place with another transition, all transitions that share this input place are enabled, as they have the same set of input places. This allows making a choice freely, on which transition to fire.

The above notion of free-choice has been introduced as *extended free-choice* in literature [33]. The original definition of the free-choice property requires that if two transitions share an input place, this is their only input place [111]. This restriction can be relaxed, as classical free-choice nets and extended free-choice nets are behaviorally equivalent [33]. For a comprehensive discussion of free-choice Petri nets and their properties, we refer to [68].

Workflow nets and free-choice nets are Petri net classes with certain structural properties that facilitate their analysis. In the following, we emphasize behavioral properties that have been proposed for verification, i. e., enforce the absence of certain behavioral anomalies.

The tokens of a net system can be understood as information, e. g., data objects or controlling data, used to steer the enactment of processes. The case that a process gets stuck prior to proper termination, as insufficient tokens are available to fire a transition, is referred to as a deadlock. Inversely, if tokens remain in the net after its termination, this indicates that some information has been produced redundantly, i. e., has no value for the process.

Boundedness

In particular, a net system that is able to produce an unlimited amount of tokens is commonly seen as behaviorally erroneous [282]. As a consequence of unlimited tokens, the state space of the net system grows infinitely and verification of certain properties may not be decidable any more [112]. Figure 11 shows a net system that is not bounded. Firing of C puts a token on the final place and enables B, which in turn enables C when firing, and hence produces an infinite number of tokens in the final place. The *boundedness* property excludes such anomalies, as it requires an upper bound of tokens that can be produced in a net system.

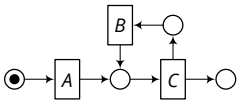


Figure 11: An unbounded net system.

Definition 3.7 (Boundedness, Safeness)**.**
A net system $S = (N, M_0)$ with $N = (P, T, F)$ and $M_0 \in \mathbb{M}$ is *bounded* iff for each place $p \in P$ there exists an upper bound $n \in \mathbb{N}$ such that $\forall\, M \in [N, M_0\rangle : M(p) \leq n$.
A net system is called *safe*, if it is bounded and the upper bound equals 1. ◂

Safeness

From the boundedness property follows that the set of reachable markings $[N, M_0\rangle$ is finite. *Safeness* is a particular kind of boundedness, as it requires that each place is upper-bounded to one token. Nevertheless, a marking of a net system may comprise several tokens.

Soundness

Further, we recall the *soundness* property that has been devised as a correctness criterion for business process models [282]. To this end, correctness is defined as a set of minimal requirements that a process definition should satisfy [283]. Soundness has been defined for workflow systems, but can be decided for business process models, if net-based formalisms are available and the nets can be transformed to workflow nets [306].

Based on the structural restriction to workflow nets, soundness requires that each case starts with exactly one token in the initial place and terminates with exactly one token in the final place. No case must be stuck, i. e., the net system has no deadlock, and termination implies that no token remains in the net, except on the final place. Furthermore, each transition must be able to successfully participate in a case of the process, as it would be unnecessary otherwise. Both net systems depicted in Fig. 10 show these properties.

Definition 3.8 (Soundness)**.**
Let $S = (N, M_i)$ be a workflow system with $N = (P, T, F)$, $i \in P$ its initial place, and $o \in P$ its final place.
M_i denotes the initial marking such that $M_i(i) = 1 \wedge \forall\, p \in (P \setminus \{i\}) : M_i(p) = 0$ and for the final marking M_o holds $M_o(o) = 1 \wedge \forall\, p \in (P \setminus \{o\}) : M_o(p) = 0$.
S is *sound*, iff

- for all reachable markings $M \in [N, M_i\rangle$ there exists a firing sequence σ that leads to the final marking, i. e., $(N, M)[\sigma\rangle(N, M_o)$,
- M_o is the only reachable state that marks o (proper termination), i. e., $\forall\, M \in [N, M_i\rangle : M \geq M_o \implies M = M_o$, and
- each transition can fire, i. e., $\forall\, t \in T\ \exists\, M \in [N, M_i\rangle : (N, M)[t\rangle$. ◂

There exist influential relationships between these properties used throughout this thesis. Soundness implies boundedness [282, Lemma 9], and sound, free-choice workflow systems are safe [283, Lemma 4]. Further, for each acyclic directed path in a sound, free-choice net system, there exists a firing sequence that executes each transition on that path in the respective order [145, Lemma 4.2], which yields a close coupling of syntax and semantics.

For various purposes, complementary soundness definitions have been proposed that relax the minimal requirements. For instance, to facilitate process modeling not each possible firing sequence must terminate properly, to verify process model interaction some transitions are allowed to be dead. A comprehensive discussion and comparison of these and further definitions are provided in [295].

3.3 BEHAVIORAL RELATIONS

The behavior expressed by business process models can be described by the traces a net system can produce or its state space. Behavioral relations capture behavioral semantics between pairs of transitions of a net system, e. g., their ordering in a trace, their mutual exclusiveness in any trace, or their concurrent enabling.

Our work builds on successor relations as defined in [314] that denote the potential co-occurrence of transitions in a trace of a net system and their execution ordering. Successor relations and their background will be discussed throughout this thesis. Therefore, we provide only their definition and a brief characterization, here. Based on [314], we recall the following definitions.

Definition 3.9 (*k*-Successor Relation)**.**
Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$, $\overline{T} \subseteq T$ a set of transitions, and $k \in \mathbb{N}, k \geq 1$.
The *k-successor relation*, $\rhd_k^\sigma \subseteq \overline{T} \times \overline{T}$, for a *trace* $\sigma \in \mathcal{T}(N, M_0)$ is defined by:

$$(x, y) \in \rhd_k^\sigma \iff \exists\, 1 \leq i \leq |\sigma| : \sigma(i) = x \ \wedge\ \sigma(i+k) = y \ \wedge\ x, y \in \overline{T}$$

The *k-successor relation*, $\rhd_k^S \subseteq \overline{T} \times \overline{T}$, for *net system* S is defined by:

$$(x, y) \in \rhd_k^S \iff \exists\, \sigma \in \mathcal{T}(N, M_0) : (x, y) \in \rhd_k^\sigma \qquad \blacktriangleleft$$

The k-successor relation captures the distance between the occurrences of a pair of transitions in a dedicated trace. For a pair of transitions $x, y \in \overline{T}$ it holds that if $(x, y) \in \rhd_k^S$ there exists a trace such that y is the k^{th} successor of x. $\overline{T} \subset T$ allows the definition of successor relations on a subset of transitions of a net system, e. g., to exclude transitions that have no business semantics.

As transitions may occur in a set of traces with different distances between them, their respective pairs can occur in various configurations of the k-successor relation induced by the parameter k. k-successor relations characterize the behavioral relations between any pair of transitions $x, y \in \overline{T}$ in a net system, i. e., if there exists no k, such that $(x, y) \in \rhd_k^S$ holds true, then these to transitions do not occur together in any trace and are, therefore, mutually exclusive.

The parameter k induces a number of different relations for a single net system. However, once a certain bound $b \in \mathbb{N}$ is reached, the relations do not change any more. That is, for all k-successor relations

with $k \geq b$ holds that $\triangleright_k^S = \triangleright_b^S$. In [314], it has been shown that the maximum value for this bound is determined by the number of transitions in the net system.

Notation 3.4 (Successor Bound). Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$. The successor bound for S is defined as $b(S) = |T|^2$. ◀

k-successor relations are the basis for the definition of an up-to-k-successor relation and a minimal-k-successor relation, either for a trace or for a system.

Definition 3.10 (Up-to-k-Successor Relation)**.**
Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$ and $k \in \mathbb{N}, k \geq 1$. The *up-to-k-successor relation*, $>_k^\sigma \subseteq T \times T$, for a *trace* $\sigma \in \mathcal{T}(N, M_0)$ is defined by:

$$(x, y) \in >_k^\sigma \iff \exists 1 \leq l \leq k : (x, y) \in \triangleright_l^\sigma$$

The *up-to-k-successor relation*, $>_k^S \subseteq T \times T$, for *net system* S is defined by:

$$(x, y) \in >_k^S \iff \exists 1 \leq l \leq k : (x, y) \in \triangleright_l^S$$ ◀

Two variants of successor relations have been introduced for the synthesis and analysis of process models before, and up-to-k-successor relations are a generalization of them by the parameter k. The *up-to-1-successor relations*, which is by definition equal to the 1-successor relation of the same net system, captures pairs of actions that can occur directly after one another in the traces of a net system and have been proposed for process mining [284]. They provide the basis for the α-algorithm that obtains such ordering relations from a set of traces of historic processes, the log, and allows deriving a net system that complies with the log [291]. Eventual successors, i. e., up-to-k-successor relations with $k \geq b(S)$, capture pairs of actions that may co-occur in a process execution, also if they do not occur directly after one another. They build the basis for behavioral profiles [319] and have been introduced for consistency management of process models [319] and process model similarity search [84, 158].

For a transition pair $(x, y) \in >_k^S$ holds that it is also contained in $>_{k+1}^S$, but may not be contained in $>_{k-1}^S$. Hence we recall a relation that determines the minimal k for which a pair is contained in the up-to-k-successor relation.

Definition 3.11 (Minimal-*k*-Successor Relation)**.**
Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$ and $k \in \mathbb{N}, k \geq 1$.
The *minimal-k-successor relation*, $\trianglerighteq_k^\sigma \subseteq T \times T$, for a *trace* $\sigma \in \mathcal{T}(N, M_0)$ is defined by:

$$(x, y) \in \trianglerighteq_k^\sigma \iff (x, y) \in \triangleright_k^\sigma \wedge (x, y) \notin \triangleright_{k+1}^\sigma$$

The *minimal-k-successor relation*, $\trianglerighteq_k^S \subseteq T \times T$, for *net system* S is defined by:

$$(x, y) \in \trianglerighteq_k^S \iff (x, y) \in \triangleright_k^S \wedge (x, y) \notin \triangleright_{k+1}^S \quad \blacktriangleleft$$

All types of successor relations can be computed from the traces of a net system. Due to cyclic dependencies, net systems may show an infinite set of traces. Still, these relations can be computed from the state space of a net system [317, 135]. However, if the system is unbounded, their computation may not be decidable [112]. For sound, free-choice workflow systems, these relations are computed in $\mathcal{O}(n^3)$ time to the number of transitions [314].

Finally, we present a concurrency relation, which denotes the enabling of two or more transitions such that these transitions do not compete for a token.

Definition 3.12 (Concurrency Relation)**.**
Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$ and $\overline{T} \subseteq T$ a set of transitions.
The *concurrency relation*, $||_c^S \subseteq \overline{T} \times \overline{T}$ for net system S is defined by:

$$\begin{aligned}(x, y) \in ||_c^S \iff\ & x, y \in \overline{T} \wedge x \neq y \wedge \exists\, M_x, M_y \in \mathbb{M}, M \in [N, M_0\rangle : \\ & (N, M_x)[x\rangle \wedge (N, M_y)[y\rangle \wedge M \geq M_x + M_y \quad \blacktriangleleft\end{aligned}$$

Here, markings M_x and M_y enable transitions x and y, respectively. $M \geq M_x + M_y$ denotes a reachable marking that enables both transitions concurrently. The concurrency relation is symmetric. Note that for a pair of transitions $x, y \in \overline{T}$, their concurrency implies symmetric successorship, i. e., $(x, y) \in ||_c^S \Longrightarrow (x, y), (y, x) \in >_k^S$ for any $k \geq 1$.

Having provided the fundamental definitions required for the formalizations in this thesis, we now briefly look at process alignments that found the basis for comparing two process models with each other.

3.4 BUSINESS PROCESS ALIGNMENTS

The term alignment has been used manifold in various fields of science and generally refers to an arrangement of objects to facilitate their comparison or establish an agreement of commonalities among them. In information systems research, an alignment refers to the identification of relations between the entities of two conceptual mod-

els [86] and emerged from the field of data and schema integration out of the need to integrate heterogeneous databases [20].

Alignments in general

By now, an abundance of application domains has adopted the underlying concepts and techniques to discover alignments [242], including, but not limited to, data warehouses that integrate heterogeneous data sources [32], service oriented computing that composes web services with heterogeneous interfaces [310, 115], and query processing where the terms used in a query may not be the same terms as specified in the underlying data [247]. In these scenarios, construction of an alignment takes place at different points in time. For the former two cases, alignments are created at design-time, i. e., to configure a system before it is used. For instance, for each input data source of a data warehouse, an alignment between the schemata of the data source and the warehouse database is established before data from that source can be incorporated into the warehouse. In contrast, query processing constructs alignments at run-time, i. e., each query needs to be aligned with the records of a database in order to decide whether the record is a match. There is a vast body of work on this topic that emerged from schema and ontology matching, e. g., [242, 262, 50, 86], and which we cannot cover here. Instead we focus on concepts and techniques to align business process models.

Aligning business process models

A *process alignment* refers to correspondences between elements of two process models, which is a prerequisite to comparing these models [316]. That is, correspondences are points of references in a model. In order to compare two process models, properties of a set of reference points and relations between them in one model are compared to properties of the set of corresponding reference points and relations between them in the second model. Such relations can be of various types, e. g., paths between two nodes in a graph, execution constraints between two actions, or associations between data objects and actions. Properties are, for instance, required resources, assigned roles, or execution time of an action.

Related work on process model search often abstracts from an explicit alignment and assumes that corresponding actions of two distinct process models are equal, which is adequate in the light of keeping the formalization of a search technique concise. On the other hand, there exists a considerable body of work that explicitly addresses the identification of correspondences among process models. For process model search in this thesis, the alignment of process models is made explicit as it is a prerequisite, for instance, to assess the similarity of two process models. However, we do not provide new techniques to discover and evaluate alignments, as this is a research topic on its own. Therefore, we will first introduce required terms and definitions. Then, we give a brief overview on the construction of process alignments to offer a starting point for further review of this topic.

Terminology & Definitions

The importance of process alignments has been stressed before, in particular in the context of comparing business processes [72, 74, 79]. The essence of a business process is the set of tasks whose execution leads to the achievement of business goals [322]. In process models, these tasks are represented by actions — nodes in the process model's graph, besides gateways, events, etc. Hence, process alignments typically resort to the alignment of actions of process models [74, 316].

We define the following terms in compliance with [86]. A *correspondence* relates actions of one process model with actions of another process model, and indicates that these actions represent the same operational concept in both business processes. The set of all correspondences between two process models is referred to as an *alignment* of these models. Matching denotes the process of discovering an alignment, which receives two input process models, compares the actions of both models, and provides a set of correspondences between the two models as an output, cf. [93]. An alignment is not required to be complete with regard to the set of actions of either model, i. e., there may exist actions in both models that remain unaligned.

Complex & uncertain correspondences

It is not always possible to identify precise correspondences between actions of business processes, due to heterogeneity inherent in process model collections, cf. Sect. 2.4. For instance, an action in one process may read "check invoice" and in another "verify data". While these actions refer to similar concepts, "check invoice" is a more specific description of a task. On the one hand, heterogeneity may lead to complex correspondences, i. e., an action of one process model corresponds to a set of actions in another process model, or a correspondence may even relate two sets of actions [316]. On the other hand, a certain degree of uncertainty may be comprised by a correspondence if the actions of two process models are not equal. Therefore, Madhavan et al. propose to "incorporate inaccurate [correspondences] and handle uncertainty" [182].

In our work, we leverage Petri net systems as a basis to compare the behavior of process models, thanks to their simple yet formal and established behavioral semantics [283]. In general, actions of a business process model are mapped to transitions, when transforming a process model into a Petri net [177]. Therefore, we define process alignments directly over the sets of transitions of a pair of Petri nets to avoid unnecessary confusion. Our definition of process alignments follows [312].

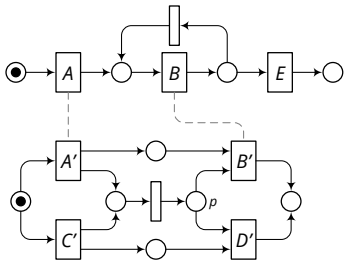


Figure 12: An alignment of example net systems of Fig. 10

Definition 3.13 (Process Alignment)**.**
Let $N = (P, T, F)$ and $N' = (P', T', F')$ be two Petri nets.

- A *correspondence relation* $\backsim \subseteq T' \times T$ associates corresponding transitions of the two nets to each other.
- A correspondence relation is called *elementary*, iff $\forall\, t', t \in T' \cup T : |\{t' | (t', t) \in \backsim\}| \leq 1 \wedge |\{t' | (t, t') \in \backsim\}| \leq 1$, and otherwise *complex*.
- A *confidence function* $\phi : \backsim \mapsto [0, 1]$ assigns confidence values between zero and one to pairs of transitions in the correspondence relation $\backsim$.
- An *alignment* is a tuple $(\backsim, \phi)$, such that $\backsim$ is a correspondence relation and ϕ is a confidence function for $\backsim$. ◂

Figure 12 visualizes an alignment of the exemplary nets introduced above, where the correspondence relation consists of two pairs of transitions, $\backsim = \{(\mathsf{A}, \mathsf{A}'), (\mathsf{B}, \mathsf{B}')\}$.

The definition of process alignments allows for complex correspondences, i. e., a transition of one model may participate several times in pairs of the correspondence relation. In theory, this allows for compensating different levels of granularity of process models. However, resolving differences in model granularity is far from trivial and only partially addressed by existing matching techniques [316]. In the related work on process model search, cf. Sect. 5.2 and Sect. 6.2, complex correspondences are, but for a few limited exceptions, not addressed at all. Since automatic matching is crucial for comparing large sets of process models in the context of search, we resort to elementary correspondences. That is, we assume that only single pairs of transitions of two models correspond to each other. The aforementioned uncertainty of a correspondence is accounted for by the confidence function that captures the degree of conformance of a pair of corresponding actions.

Elementary correspondences

We introduce a shorthand notation for correspondences, to keep formalizations and discussions in the remainder concise.

Notation 3.5. Let $\backsim \subseteq T' \times T$ be an *elementary correspondence* relation between two nets $N = (P, T, F)$ and $N' = (P', T', F')$. $\backsim_t = (t', t)$, with $t \in T, t' \in T'$, denotes a pair of corresponding transitions, i. e., $(t', t) \in \backsim$. ◂

Consequently, we quantify the confidence of a pair of corresponding transitions, $(t', t) \in \backsim$, with $\phi(\backsim_t)$.

Construction of Process Alignments

An alignment may be defined manually [75], where human experts relate actions of process models to each other. Although this promises a high quality of the identified correspondences, it is not feasible for large process model collections, and particularly not in the case of process model search. The sheer amount of models, the continuous evolution of the model collection, and the use of query models that cannot be known a priori require automatic processing.

Automatic construction of an alignment is, however, far from trivial. The reason for this can be found in the *vocabulary problem* that has been studied by Furnas et al. [92]. In exhaustive experiments, the authors unveiled that in average only a dozen of thousand people will agree on the same term for a given concept, independently of the application domain. Terminological heterogeneity, i. e., the usage of different words for the same concepts, has been explored in the field of ontology matching [86] and is generally addressed by applying a set of precise matching rules or by computing similarities over all possible correspondences and comparing them to choose the best [182]. The opposite problem, referred to as semiotic heterogeneity, that is manifested in different interpretations of the same term [220] is a hard problem that requires large textual contexts, which is typically not available in the labels of business process models.

Precise matching

Approaches to *precise matching* rules for business process alignments consider ontological annotations of actions, cf. [38, 81, 193, 26, 61, 265]. In these cases, each action is characterized by one or several attributes of a common business process ontology and matching of actions is conducted by reasoning on the relations between the ontological annotations. Obviously, this requires the definition of the respective concepts for each stored process model and the query beforehand and cannot be done automatically in the general case.

Similarity matching

Thus, the name, or label, of actions in the process model is used [79] and an alignment is constructed by *similarity* computation. For this purpose, Gal and Sagi [93] propose a multistage approach that uses two different kinds of matchers, i. e., algorithms that conduct the matching process. First line matchers take a pair of models as input and compute a similarity matrix over the Cartesian product of elements, here subsets of transitions of both process models. That is, first line matchers operate directly on the semantics of the business process or net system, respectively. Contrarily, second line matchers

only take one or several similarity matrices as input and transform this into another similarity matrix. Application of second line matchers eventually leads to an alignment, as they exclude all undesired correspondences. Typically, these second line matchers solve the *assignment problem*, to which the Hungarian Method and Stable Marriage algorithms are prominent solutions, cf. [43].

Second line matchers often produce a binary similarity matrix [252], where entries that have a similarity value of 1 are denoted correspondences and entries with a value of 0 are not. Following [182] and [252], the degree of uncertainty or similarity of matches should however be preserved. Therefore, the confidence function introduced in Def. 3.13 can provide the similarity that has been disclosed by first line matchers.

Syntactic label similarity

Solutions to bridge terminological heterogeneity, can be classified into syntactic and linguistic[1] approaches. The majority of approaches toward process model search applies *syntactic* approaches. For this purpose, a large body of string-based comparison techniques exists, cf. [219, 55], for instance, string equality, substring containment, n-grams [5, 86], or edit distances for labels. For the latter, the renowned Levenshtein distance [168] computes the minimal number of operations (insert, delete, update) required to transform one label into another. Some approaches first tokenize a label before assessing similarity, which allows representing labels as vectors and compute their similarity with the Cosine similarity [256].

Linguistic label similarity

All of these approaches are affected by *linguistic phenomena*, e. g., synonyms and homonyms, and are investigated by natural language processing [187]. For instance, the elimination of stop words, word-stemming [237], and the application of dictionaries [210] or domain-specific linguistic models [186] have been used to identify correspondences, cf. [81, 300, 75, 167].

As a preliminary step to either approach, terms of labels may be preprocessed, e. g., characters are set to lower case, and blanks and special characters may be replaced. In that context, [221] proposes replacing terms globally prior to comparing labels to overcome the vocabulary problem stated above.

Contextual similarity

First line matchers are not restricted to comparing the labels of actions of business processes to determine their similarity, but can incorporate the *context of these actions*. For this purpose [81, 222] also incorporate the structural context in terms of predecessors and successors in the process model graph into the similarity assessment of actions. In [73], the alignment is iteratively computed such that a maximum value for the structural similarity of two complete process model graphs is achieved, in order to obtain an optimal alignment.

1 Some authors refer to linguistic solutions by the term semantic, e. g., [300], which in turn is used for approaches that use ontological annotations.

Part II

SEARCHING PROCESS MODELS BY EXAMPLE

4

BEHAVIORAL DISTANCES TO SEARCH PROCESS MODELS

This chapter is based on results published in [12, 158, 154, 110, 156].

Searching process models by example assumes a regular business process model as a query and provides answers that comply with this process model. In order to obtain such answers, a search technique must first extract the features by which a comparison of query and candidates is conducted in the processing of the search question.

In this chapter, we introduce the formal framework for searching process models based on successor relations as an abstraction of their behavior. A large body of research exists to capture and compare the behavior of business processes, on which we elaborate in Sect. 4.1. Based on the presented concepts, we discuss how commonalities in the behavior of process models can be identified by successor relations in Sect. 4.2. The quantification of commonalities and differences leads to distance functions that found the basis for search, cf. Sect. 4.3.

To support the comprehension of search results and facilitate reuse of matched models, we introduce a set of quality measures in Sect. 4.4 and show how common behavior can be projected on the process model's graphical representation in Sect. 4.5.

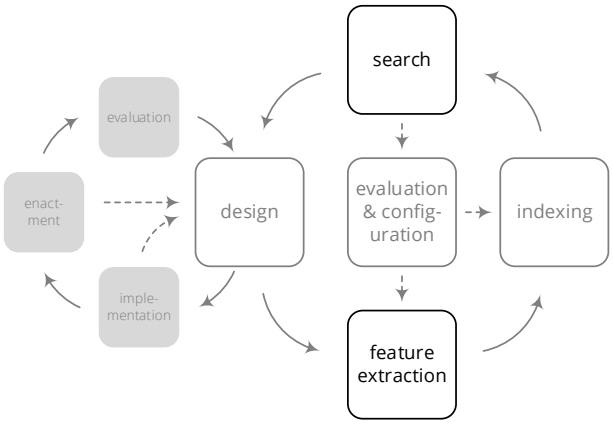


Figure 13: Topics addressed in this chapter: Extraction of features, foundations for search, and presentation of search results.

4.1 EXPLORING THE BEHAVIOR OF BUSINESS PROCESSES

Distance and similarity are closely related terms that address the commonalities between two systems or models with respect to certain properties. A large body of work, in particular in the area of information retrieval [188, 16], has addressed the semantics of similarity and distance, and techniques to measure them. For instance, Lin [171] elaborates on the information-theoretic definition of the term similarity independent of a particular domain and proposes that similarity is a function of the amount of information of the commonalities and differences of two compared entities. This definition aligns well with many general-purpose similarity measures, of which a comprehensive survey is provided in McGill et al. [196]. More recent research on similarity and distance can be found in [86, 188, 16]. Due to its broad coverage in literature, we do not examine this topic in more detail hereafter and refer to the aforementioned work. Instead, we focus on methods to encompass, explore, and compare the behavior of dynamic systems and business processes in particular, as a basis to determine commonalities among them.

Behavioral Semantics for Business Processes

In order to compare business processes, humans typically resort to the graphical structure of the process model, that is, the set of common nodes and connections between them [66]. However, the purpose of a process model is to prescribe the execution of actions with respect to their environment [322]. Therefore, we argue that the essence of a process model's semantics is its behavior. Nevertheless, we acknowledge that the behavior is expressed by the model's graph structure plus the semantics of the elements that form this graph. Process models with different graph structures may express the same behavior, whereas models that show considerably similar structures may, in turn, yield completely different behavior, cf. Sect. 2.4. This has been acknowledged by researchers of business process model management early, and means to capture and compare the behavior of process models have been established.

The majority of approaches to capture the behavior of process models for comparison disregards of properties of concurrency and real-time or stochastic aspects, and focuses on the actions carried out as a means to advance the process. Furthermore, the nature of these actions is typically neglected and they are represented by their inscriptions — uninterpreted symbols referred to as labels [303]. Such approaches are generally subsumed under the term *labeled transition systems*.

State space, Automata

The most generic representation of the behavior of a discrete, dynamic system is its *state space*. A state represents the system with a certain configuration of all of its relevant properties. Changing one

of these properties may change the state of the system, captured by state transitions. The set of states reachable through state transitions starting from some initial state is called the state space. Automata are used as a means to prescribe the behavior by restricting possible state transitions for certain states and go back to Huffman [127], Mealy [198], and Moore [216]. In the context of analyzing business processes, state transitions typically represent actions of the process and are associated with the action's label. Traversing a sequence of state transitions in the state space or automaton leads to sequences of labels. A sequence that starts in an initial state is called a trace.

Trace semantics

The concept of traces to describe processes has been introduced by Hoare's seminal work on Communicating Sequential Processes [123]. *Trace semantics* are based on the assumption that the behavior of a process can be described by a possibly infinite set of traces, i. e., sequences of action labels. In [125], Hoare proposes a complete process algebra that provides many useful properties and operations to capture and compose business processes. Trace semantics are purely sequential and concurrent actions can be captured as interleaved labels in a trace [125].

Process calculi

Trace semantics have received criticism as they are not capable of capturing the "moment of choice" in the case that a process may choose from several alternative actions at some point during its execution [120]. As a response, Milner proposed a *Calculus for Communicating Systems* [211] that captures the behavior of processes by a set of axioms and their composition rules. This has been adopted and led, for example, to the development of the well-known π-Calculus [212].

Petri nets

Petri nets, devised by Petri in his PhD thesis [232], are a type of labeled transition system that has received wide recognition and application, in particular, for the analysis of behavioral aspects of process models [283]. Petri nets provide rather simple yet concise sequential execution semantics, where transitions represent actions, or, more precisely, an event that demarcates a state change in the life cycle of an action, cf. [321]. The syntax and semantics of Petri net systems, cf. Sect. 3.2, are apt to describe processes and provide semantics for traces and calculi. States are represented by the marking of places; enabling and firing semantics of transitions yield properties of process calculi, and firing of transitions leads to firing sequences and traces.

Based on these diverse process semantics, researchers have developed a spectrum of equivalence notions for workflows and business processes [301], and showed that these can be applied to Petri net systems [236]. However, it has been emphasized that pure equivalence notions are not sufficient in order to compare and search for process models, e. g., in [66, 300, 75]. We elaborate on equivalence definitions and approaches to compare the behavior of business processes in the following.

Comparing Business Processes by Traces

Trace equivalence

Hoare defines a process "by the set of all traces of its possible behavior" [124, 125]. Consequently, two processes are considered to be equivalent, if they show the same sequences of observable actions [125], generally referred to as *trace equivalent*. As mentioned above, concurrent actions of a process model are captured by interleaving labels. Concurrent actions that are overlapping during their execution can be expressed more precisely by expanding each action into two events that represent their beginning and termination, respectively, and capturing them in an interleaving order in traces [120]. Trace equivalence has been used, among others, as a criterion to decide whether process refactoring operations preserve the behavior of a process [311], or to compute firing sequences that differentiate one process model from another to understand their evolution over time [190].

It has been acknowledged that two business processes may not offer exactly the same sets of actions. By means of abstraction, such actions are considered internal and are hidden from the traces a process can produce [125]. As a result, trace equivalence holds, if all observable traces of two processes are equal. This kind of abstraction has been adopted by Li et al. [170], where trace projections reduce the traces of one process to actions it has in common with a second process. Based on trace projections, the authors denote processes to be similar, if for each trace of one process there exists an equivalent trace of the second process.

Following Hoare's argumentation of characterizing a process by the sets of observable traces, distance measures have been proposed that evaluate commonalities of the sets of traces of processes. In that context, Medeiros et al. propose rather simplistic measures that judge on the ratio of common traces of a pair of processes by all traces of either process [66]. Mahleko et al. [185] consider business processes to be similar, if they share at least one common trace. Gerke et al. [97] quantify process similarity by longest common subsequences compared to the length of corresponding traces. A survey of algorithms to compute longest common subsequences is provided in [29]. Alternatively, the string edit distance [40] has been used to measure similarity of traces in [271].

Approximating infinite traces

If sets of traces are computed from process models to capture their behavior, they may become infinite due to loops in the process, i. e., unbounded repetition of the same subsequences. This issue has been addressed by researchers by an approximation of the number of iterations that are allowed for a loop. In [185], duplicate labels are removed from sequences, i. e., no iterations are captured at all. In contrast, in [97] the number of iterations is restricted by a nondescript variable parameter. Such a parameter could be derived from the average number of iterations that have been observed in historical cases

of the process [271]. Another approach, proposed by Wombacher et al. [323, 324], splits traces into sets of n-grams, i. e., unique subsequences consisting of n labels each [86]. For any process with a finite set of labels, n-gram representations are finite as well. Wang et al. [307] explicitly capture those parts of the behavior that are part of a loop by so-called principal transition sequences, i. e., the set of traces is decomposed into iterative and non-iterative subsequences, which requires the analysis of a process' state space. These sequences are then used to compare and quantify behavioral commonalities.

Comparing Business Processes by States

Bisimulation

The criticism directed at trace equivalence not being able to capture the "moment of choice" obviously applies to any approach that compares process models by their sets of traces. In response, Park [230] and Milner [211] independently proposed the concept of *bisimulation*, which examines whether two processes are able to simulate each other. In addition to the requirement to produce equivalent traces, bisimulation equivalence also requires that if one process offers a choice, i. e., a set of alternative actions to execute, at a certain point during its execution, any equivalent process must offer the same set of alternatives at the same point. This idea is bound to the concept of states, i. e., two processes reach a state after a given firing sequence. If the state of one process offers the same choices as the state of the other process, the pair of states is considered equivalent. Bisimulation coincides with the requirement that for each state of either process, there exists an equivalent state of the other process [211].

The problem of business processes having unequal sets of actions has been tackled by the concept of observability. If an action cannot be observed, i. e., is not matched by an action of another process, it is considered hidden. Consequently a spectrum of bisimulation equivalences has been proposed, cf. [301]. *Observation equivalence*, also referred to as weak bisimulation, cf. [211, 303, 120], requires that if one process offers a set of alternative actions, any equivalent process must offer the same set of choices after a possibly empty set of unobservable actions [211]. Branching bisimulation equivalence, introduced by Glabbeek [302], is the strongest observation equivalence [120] and requires that the same set of choices is offered before and after each unobservable action. Van der Aalst and Basten [286, 287] proposed the concept of workflow inheritance, which addresses the question, whether one process is a subclass of another. Under certain types of abstraction, i. e., hiding or blocking actions, inheritance is decided by means of branching bisimulation. Closely related is the concept of stuttering equivalences, presented by Baier and Katoen [17], that does not only allow for hiding but also repeating actions in one process which would not be repeated in the second.

Workflow inheritance

Again, equivalences are not generally adequate to compare the behavior of business processes, as they provide only a binary answer and cannot provide a measure of the behavior they share. Therefore, Sokolsky et al. [267] replay processes to measure the relative amount of how much the processes are able to simulate each other, which the authors refer to as approximate quantitative bisimulation. If, at any state, one process does not offer the same choices as the other the similarity of these states is less than 1. The overall similarity of processes is computed iteratively such that differences in succeeding states of a process have less influence. A rather similar approach has been followed by Nejati et al. [221], where the similarity of processes is computed in an iterative fashion by the similarity of states. In each iteration, the similarity of a pair of states is computed from their context, i. e., the similarity of their succeeding and preceding states of the previous iteration. Starting from an initial similarity, this is repeated until the result converges.

We emphasize that the small number of approaches that incorporate state spaces by means of bisimulation, compared to trace semantics, indicates that this is a criterion too strict for process model search and rather serves the purpose of comparing processes for analysis.

Comparing Business Processes by Behavioral Relations

Process semantics based on traces or states suffer from the state explosion problem [280]. That is, concurrent paths in a process model yield a large number of possible firing sequences with interleaved firing of transitions, which leads to an exponential growth of the state space. The runtime complexity of above equivalences is at least linear to the size of the state space [280], and, therefore, also exponential to the number of actions of a process. Distance measures are subject to the same issue. To tackle this problem, researchers resorted to preprocess business processes and capture their behavior in terms of abstractions — finite relations defined on the set of unique actions of a business process.

Causal footprints

Causal footprints have been introduced in order to reason on the correctness of process models [298]. A causal footprint is a set of conditions with respect to causal relations of actions that are required to hold in a process model. Footprints constrain the execution order of actions by means of look-ahead and look-back links for each action of the process. Each look-ahead link relates an action x with a set of actions, such that the execution of x leads to the execution of at least one of the actions contained in the linked set. Analogously, a look-back link relates an action x with a set of actions of which at least one must be executed as a prerequisite to x. These conditions are an abstraction of the process behavior by means of trace semantics, and, as a result, there may exist processes with different behavior that have identical footprints [298].

Besides its capabilities for efficient verification of business processes toward certain correctness properties, Dongen at al. proposed to leverage causal footprints to quantify the behavioral similarity of business processes by representing a model's footprint as a vector [205, 300]. Similarity of two footprints is computed by means of the cosine-similarity [255] of their respective footprint vectors.

Precedence relation

A more compact type of behavioral relations has been introduced in the context of workflow mining [195], that is, the discovery of process models from historic event logs that document the execution of process instances, cf. [284]. The α-algorithm [291] derives a successor relation from a given log, such that it contains pairs of actions that can be executed directly one after another. These order relations are called *precedence relations*, succession relations in [195], or transition adjacency relations in [334, 136]. From precedence relations, Petri nets can be constructed.

Vice versa, it is possible to derive precedence relations from process models by examining the traces a process can produce, cf. Sect. 3.3. Based on this idea, Zha et al. [334] and Jin et al. [136] compute the well-known Jaccard coefficient [132] from the precedence relations of two processes, which yields a normalized similarity measure between 0 and 1, where a value of 0 indicates that the processes have no commonalities, and 1 denotes that the relations are identical. In [334], precedence relations are generally computed from the state space of a Petri net system, while the authors propose some reduction techniques restricted to well-structured workflow nets that mitigate the state explosion problem by decomposing the net and reducing concurrency in the examined state spaces. The authors of [136] propose an approach based on the complete prefix unfolding of a net system [197, 85] that shows better efficiency also for unstructured net systems.

Weak order relation

An extension of the precedence relation by means of transitivity is presented by Eshuis and Grefen [84] and Weidlich et al. [315], who define a successor relation which captures all pairs of actions in a process that can be executed one after another eventually in a process instance. Similarity is computed by the Jaccard coefficient of the order relations of two processes. In [84] the relation is referred to as "before relation" and is computed from BPEL process definitions, i. e., structured process models. In contrast, Weidlich et al. [315] define the *weak order relation* based on trace semantics of a process. The authors argue that weak order relations are well suited to judge on the consistency between process models of different granularity. The weak order relation can be computed for bounded net systems [317, 135] and their efficient computation has been shown for sound, free-choice workflow systems [319].

Successor relations

In later work by Weidlich and van der Werf [314], precedence and weak order relations have been generalized by the concept of parame-

terized relations that allow for a variable look-ahead, called *successor relations*. By that definition, precedence is a 1-successor relation and weak order the union of k-successor relations for any $k \in \mathbb{N}, k \geq 1$. Based on this definition, up-to-k-successor and minimal-k-successor relations are defined, cf. Sect. 3.3.

From the elementary relations, i. e., precedence, weak order, or in general k-successor, more expressive relations can be derived that distinguish the relationship between two actions more precisely [291, 1, 315, 314]: exclusiveness, strict order, and interleaving order. Following [314], we refer to these relations as *relation sets*, hereafter. Exclusiveness denotes that two transitions do not appear in an elementary relation. In case of weak order, this implies that these actions can never be executed in the same process instance. Strict order denotes that two actions are always executed in the same order, if they occur together in a process instance. In contrast, interleaving order allows executing a pair of transitions in either order. Based on this notion, process model similarity measures have been defined that provide configurable weights to incorporate these relation types in a combined distance measure [1, 158]. Jin et al. [135] search for process models by requiring that a given relation set must be mirrored by a matched model.

All successor relations, and approaches to compare process models based on them, have in common that they only address the execution order of actions in some process instance; they do not account for causal relations between actions. Therefore, Weidlich et al. [318] introduce the *co-occurrence relation* that comprises all pairs of actions, such that, for each pair it holds that, if one action is executed in a process instance, the second action must be executed as well. This notion does not capture the ordering of the execution of actions. Although this notion acknowledges causal relationships it is still not as expressive as behavioral footprints with respect to causality. The authors show that the causal behavioral profile, i. e., the behavioral profile plus the co-occurrence relation, can be computed efficiently [318].

Discussion

The contributions of this thesis are based on comparing the behavior of business process models by their successor relations. In earlier work [159, 158, 154], we have been among the first to leverage successor relations as a means to compare process models for search. Recently, a large body of research has addressed the topic of process model search with regard to querying [308] and similarity search [22], the majority of which we discussed above. Among these works are approaches that also resort to successor relations for assessing the relevance of a model for a query and deciding a match. We discuss these works with respect to the search techniques discussed in this thesis in detail in Sect. 5.2 and Sect. 6.2.

Process alignments

With regard to process alignments, cf. Sect. 3.4, related work generally assumes actions of process models to be sets that share common actions. The intersection of the sets of actions provides the process alignment. Although this work abstracts from the problem of constructing an alignment, we explicitly incorporate it into the techniques to compare process models. On the one hand, this allows the definition of process model similarity and query matching independent of a particular alignment technique, cf. [316]. On the other hand, it allows incorporating the confidence of an alignment to compare process models.

Presentation of search results

Scientific contributions to process model search have in common that they present approaches to compare process models, based on text, the model's graph structure, or behavior, assess their similarity, and decide a match. However, we observed barely any work that approached the presentation of a search result, e. g., evaluating its quality in terms of the confidence about the relevance of results and the ambiguity of a ranking. Also, reusing process models that have been found as a search result is largely neglected, as it is a step that follows upon searching for process models. Nevertheless, we argue that supporting users in understanding the search result and facilitating their reuse, e. g., to extract a fragment of a match and revise it, is crucial for making process model search effective.

In the sections that follow, we establish a foundational framework to compare process models by their corresponding successor relations, cf. Sect. 4.2, explain how behavioral distances are computed from identifying commonalities in the successor relations of a query and a candidate process model, and elaborate on their properties in Sect. 4.3. As behavioral distances are used for search, we propose a set of measures to assess the quality search result based on these distances in Sect. 4.4. Finally, in Sect. 4.5, we propose a projection that allows visualizing successor relations in the graphical structure of a business process model. This is used to highlight the commonalities between a match and a query, to support the user in comprehending the search result. Once fragments of a process model are highlighted, they can be easily extracted for reuse.

4.2 COMPARING PROCESS MODELS BY SUCCESSOR RELATIONS

In Sect. 2.4, we elaborated on a set of requirements to search process models and emphasized, among others, an *intuitive specification of the search question* and a *comprehensive discovery model* that are closely related to the skills of users. With our approach, we aim at non-expert users with respect to process modeling, e. g., business analysts and process participants, that can understand business process models and are able to express themselves through process models. These users may, however, not be capable of formulating search questions

by means of formal specifications. Instead, we propose searching for process models by giving an example of what the user is searching for. This has been inspired by Zloof, who proposed *Query by Example* for relational databases, where *'the user basically formulates his query [...] with an example of a possible answer'* [337]. Similarly, the approach presented here takes a regular process model as an example and discovers all process models that meet the query with respect to certain behavioral aspects. Matching is carried out using the notion of distance, which is discussed in more detail in Sect. 4.3. The example by which search is conducted is referred to as *query* and any process model from the repository that is compared with the query as *candidate*, hereafter. If the candidate meets the specified requirements, we call it a *match*.

We advocate that traces are a suitable representation of the observable behavior of a process, as a set of processes that perform the same actions in the same order have the same effect, independently of their internal state or decision routing. The utilization of trace semantics for comparing process models is supported by a number of approaches to compare process model behavior by trace semantics and abstractions thereof, cf. Sect. 4.1. Since the set of traces suffers from exponential growth and may be infinite [280], we rely on an abstraction provided by successor relations, formally defined in Sect. 3.3. Informally, any two actions of a process that can be executed in one process instance contribute to a successor relation. More precisely, successor relations are defined on subsequences of traces of a Petri net system. If two transitions appear in a subsequence of any trace that can be produced by the net system, they will contribute to the successor relations of that net system.

Framework overview

Figure 14 visualizes the formal background of successor relations and the conceptual framework for searching process models by example. The query provided by a user and the set of candidates stored in a process model repository are regular process models, hence from the perspective of users business process models are compared. From these models successor relations are derived, either by computation with net-based formalisms or by the traces that can be produced from the process models. A query is then actually compared with candidates based on successor relations that are abstractions over the models' execution traces. In order to treat different modeling languages uniformly in our framework, as it has been required in Sect. 2.4, we leverage existing formalisms to transform process models to Petri net systems, from which behavioral relations are computed. In the remainder of this section, we elaborate on the inference of successor relations from process models, how behavioral commonalities can be identified by comparing successor relations of a query and a candidate, and which benefits this provides for process model search.

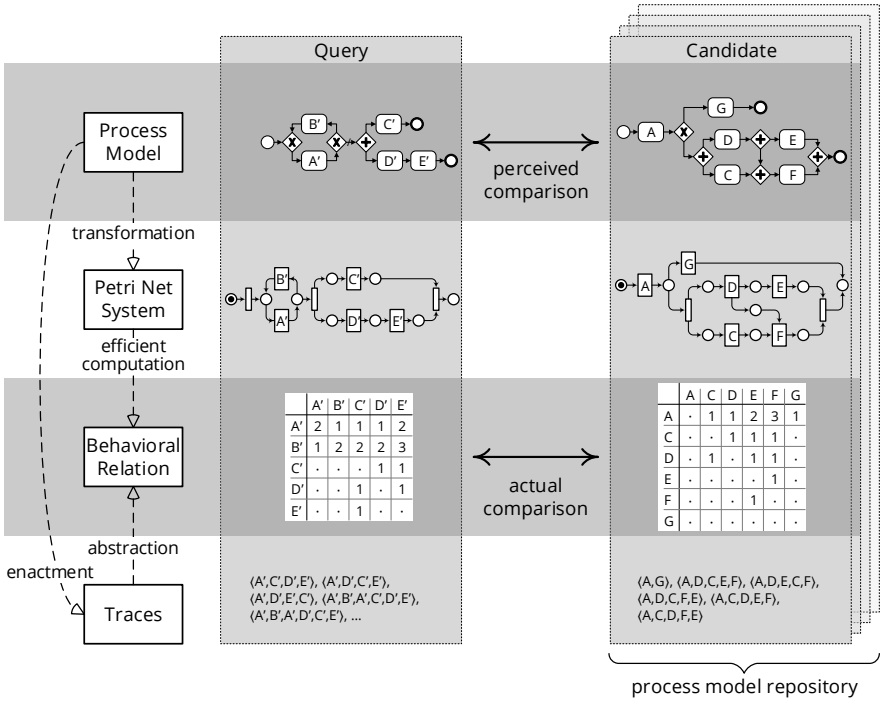


Figure 14: Comparing process models by behavioral relations.

From Process Models to Behavioral Relations

The behavior of business processes — "a series of actions that are conducted in a coordinated fashion", cf. Def. 2.1 — is exposed by their enactment, i. e., the execution of actions over the course of time, whereas the process model prescribes the order in which these actions may be executed. A plethora of modeling languages of different kinds [209] exist, of which some have received more frequent adoption than others [64], for example, the Business Process Model and Notation (BPMN) [225], Event-Driven Process Chains (EPC) [142], UML Activity Diagrams (UML AD) [106], and the Web Services Business Process Execution Language (BPEL) [89]. These approaches share the concept of control flow, i. e., they explicitly describe execution order and conditions. Other approaches, such as case handling [292], declarative process model specifications [231, 296], or artifact-centric approaches [224, 56] dictate dependencies of actions and result in a more flexible execution ordering. Nevertheless, their execution produces traces of observable actions, which allows capturing their behavior by successor relations as well.

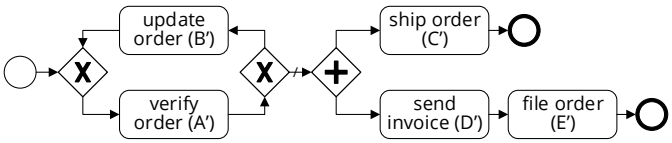
Process models to Petri net systems

Petri net systems, cf. Def. 3.2, provide a formal foundation for the semantics of business processes [283] which allows analyzing various aspects and properties on a common grounding. In business process management practice, however, high level modeling language have received higher acceptance than languages with precise formal groundings [177]. Researchers have addressed this issue and provided Petri net based formalisms for these languages, i. e., proposed

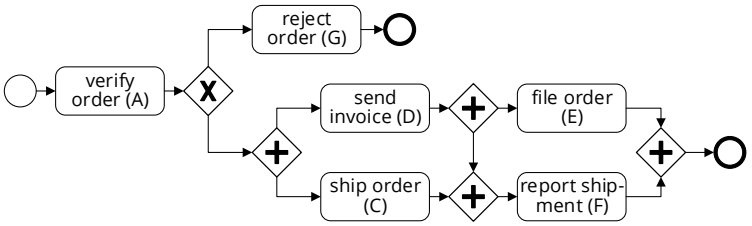
transformations to Petri net systems, in particular for their analysis, for BPMN [241, 272, 76], EPC [299, 144], UML AD [273, 270], and BPEL [121, 227, 176]. Due to the lack of precise execution semantics inherent to high level process modeling languages, not all features can be transferred to Petri net systems. For instance, execution semantics of BPMN are described comprehensively in [225], yet only in prose text rather than on a mathematical foundation. In contrast, BPEL aims at the enactment of business processes and is, therefore, formalized closely to computer programing languages. Consequently, a feature complete Petri net formalism for BPEL can be provided [176]. Nevertheless, common features have been addressed in these formalisms and implementations for an automatic transformation are available [297, 177].

Translation of business process models to Petri net syntax may lead to different classes of nets, cf. Sect. 3.2. To facilitate their analysis, these nets may be normalized. To this end, Vanhatalo et al. formulated a set of techniques to refactor and complete Petri nets, and derive workflow nets [306]. As the transformations may add additional places and transitions that have no representation in the original process model and increase the state space of the net system, Murata presents reduction rules that mitigate this issue [218]. To arrive at net systems, an initial marking of the net is required. Therefore, Decker et al. [67] analyze the instantiation semantics for above process modeling languages. Consequently, an initial marking of a net system can be established, which enables transitions corresponding with the actions or events that contribute to the instantiation of the business process.

We leverage these formalisms to capture the execution semantics of both query and candidate process models uniformly by Petri net systems, illustrated in Fig. 14. The transformation of process models to the common semantics of Petri net systems abstracts from the actual modeling language and allows comparing process models of different modeling languages. This is in line with the requirement for process model search raised in Sect. 2.4 that requires *abstraction* from irrelevant details, such as the process modeling language. Figure 15 recapitulates the exemplary order handling processes that have been outlined in Fig. 14. In these processes an order is first verified and then both goods and invoice are sent to the customer. In Fig. 15a, if verification of the order failed, it can be updated, whereas it would be rejected in Fig. 15b. The transformation of these processes to workflow systems is depicted in Fig. 16a and Fig. 16b, respectively.

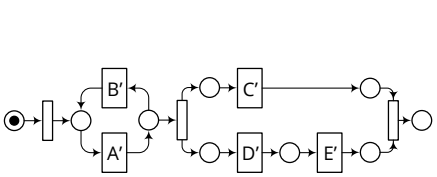


(a) Order handling process query.

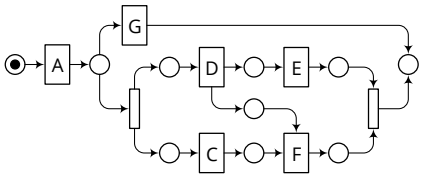


(b) Order handling process candidate.

Figure 15: Example BPMN process models.



(a) Order handling process query.



(b) Order handling process candidate.

Figure 16: Example process models transformed to Petri net systems.

Petri net systems to behavioral relations

Whereas behavioral relations are defined on the traces of a system, they can be computed directly from the Petri net systems in a finite manner, given the net systems are bounded, i. e., have a finite state space [317, 314, 136]. For sound, free-choice workflow systems, these relations can be computed efficiently from the net's structure in cubic time to the number of transitions [318, 319]. Soundness, cf. Def. 3.8, is a correctness criterion that guarantees proper termination of any process and the absence of various behavioral anomalies [282]; free-choice, cf. Def. 3.6, is a restriction on the model's structure [68].

Process models to traces

The set of traces that describes the observable behavior of a business process can be computed by executing the business process, e. g., through simulation, exploring the space of reachable states, and deriving all possible traces, cf. Def. 3.4. The traces producible from the process models of the example are shown in the bottom row in Fig. 14. From these traces, behavioral relations can be computed by inspecting, which actions are successors of which other actions, cf. Def. 3.9. Please note that the query model contains a loop, which leads to an infinite set of traces by iterating the loop arbitrary often. This issue has been addressed in related work by limiting the number of iterations [97], see also Sect. 4.1. However, the question, after how

many iterations a loop should be discarded, remains open in absence of historic execution data [271].

Once the behavioral relations of a pair of process models are computed, they are compared to identify commonalities in the behavior of the query and a candidate model. This is highlighted in the third row of Fig. 14 that presents the behavioral relations of the net systems in the form of a matrix.

Identifying Behavioral Commonalities of Business Processes

When comparing process models, both commonalities and differences can be taken into account, while these two concepts are not disjoint. For example, the concurrent execution of two actions can be considered different from a sequential execution of the same actions, as in the latter case the second action is causally depending on the first. At the same time, interleaved execution of the concurrent actions may resemble the sequence in some traces. Hence, there exist also commonalities. Discussions in literature address various kinds of differences and commonalities of process models, cf. [72, 161, 189], generally, to find similar processes and resolve differences among them. Workflow patterns explicitly aim at particular behavioral patterns of process models and allow for their comparison [290]. In the following, we examine properties of process model behavior considered in the literature and discuss how they are addressed by successor relations.

Successor relations

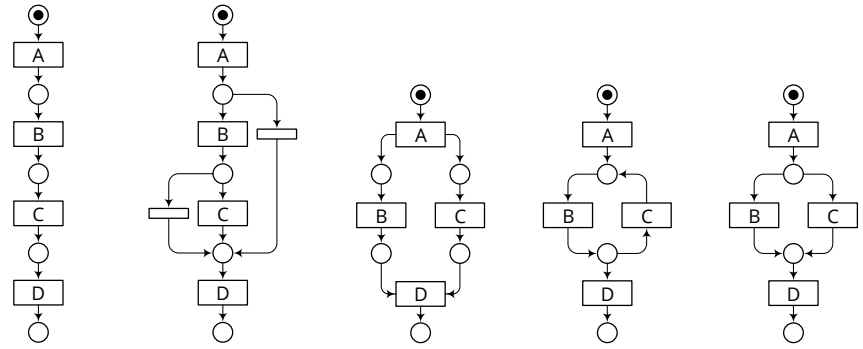
A successor relation provides a behavioral abstraction which focuses on the potential execution order of actions and neglects other behavioral details. It consists of pairs of actions (x, y) that can be executed in a certain order, i. e., a predecessor action x and a successor action y. Findings from experiments with process variants [319] and empirical work on the perception of consistency between process models by process analysts [313] suggest that successor relations capture a significant share of the behavioral characteristics of process models. Following the above presumption of comparing processes by the behavior induced from their corresponding net systems, we discuss the identification of commonalities based on exemplary net systems, depicted in Fig. 17. For the sake of comprehension, the up-to-k-successor relation [314], $>_k^S$, is repeated from Def. 3.10 in independent terms.

Definition (Up-to-k-Successor Relation, restated from Def. 3.10)**.**
Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$, and $\overline{T} \subseteq T$ a set of transitions.

The *up-to-k-successor relation*, $>_k^S \subseteq \overline{T} \times \overline{T}$, for system S contains all pairs of transitions (x, y), such that there exists a firing sequence $\sigma = \langle t_1, ..., t_n \rangle$ enabled in a reachable marking $M \in [N, M_0\rangle$, i. e., $(N, M)[\sigma\rangle$, and $i \in \mathbb{N}$, $1 < i \leq k$ for which holds $t_1 = x$ and $t_i = y$. ◂

The parameter k indicates a look-ahead, i. e., $k+1$ denotes the length of subsequences of traces. Within the bounds of these subsequences, successor relations are sought. For example, if $k = 1$, only subsequences of two actions are considered, which yields direct successors, i. e., the precedence relation. Since up-to-1-successor and 1-successor coincide, we use these terms synonymous with the term precedence relation. If the system, for which a successor relation is defined, is clear from the context, we resort to $>_k$ for simplicity, hereafter.

Table 1 visualizes the up-to-k-successor relations for the net systems depicted in Fig. 17. The sequential net system in Fig. 17a, has only one trace $\sigma = \langle \mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D} \rangle$. A look-ahead of 1 yields $>_1 = \{(\mathsf{A}, \mathsf{B}), (\mathsf{B}, \mathsf{C}), (\mathsf{C}, \mathsf{D})\}$, cf. Tab. 1a. Increasing the look-ahead k will preserve existing relation pairs while adding more relations, as longer subsequences contain also shorter ones. Therefore, it holds in a net system for all $j, k \in \mathbb{N}$ with $j \leq k$ that $>_j \subseteq >_k$. The matrices in Tab. 1 depict up-to-k-successor relations for various k, such that for a given k all fields that have a number less or equal to k indicate relation pairs of $>_k$. For instance, in Tab. 1a $>_2 = >_1 \cup \{(\mathsf{A}, \mathsf{C}), (\mathsf{B}, \mathsf{D})\}$; $>_3 = >_2 \cup \{(\mathsf{A}, \mathsf{D})\}$.



(a) Order of actions. (b) Optionality. (c) Concurrency. (d) Iteration. (e) Choice.

Figure 17: Net systems that illustrate various properties of behavioral relations to identify commonalities and differences.

	A	B	C	D
A	·	1	2	3
B	·	·	1	2
C	·	·	·	1
D	·	·	·	·

(a) Order of actions.

	A	B	C	D
A	·	1	2	1
B	·	·	1	1
C	·	·	·	1
D	·	·	·	·

(b) Optionality.

	A	B	C	D
A	·	1	1	3
B	·	·	1	1
C	·	1	·	1
D	·	·	·	·

(c) Concurrency.

	A	B	C	D
A	·	1	2	2
B	·	2	1	1
C	·	1	2	2
D	·	·	·	·

(d) Iteration.

	A	B	C	D
A	·	1	1	2
B	·	·	·	1
C	·	·	·	1
D	·	·	·	·

(e) Choice.

Table 1: Up-to-k-successor relations for net systems of Fig. 17.

Order of actions

Effects from differences in the order of actions, cf. [72, 161, 189], are lessened by a look-ahead that is larger than the number of disordered actions. If, for instance, B and C were switched in Fig. 17a, the system's 1-successor relation would be completely different, whereas the additional relations added by $>_2$ and $>_3$ would not be affected. Weak order, discussed in Sect. 4.1, coincides with $k = \infty$. However, it has been shown that $>_k^S$ of a systems S does not change anymore for any $k \geq b(S)$ [314], given $b(S) = |T|^2$ with T being the number of transitions of a net, cf. Not. 3.4. Therefore, $>_{b(S)}^S$ refers to the weak order relation, hereafter. These observations suggest that the parameter k can be used to control the sensitivity of identifying commonalities of behavioral relations.

Optionality

Comparing the behavior of two processes, a skipped action indicates that one process executes an action that has no correspondence in another process [72]. In successor relations, this is revealed by missing relation pairs that involve this action, while other pairs are not affected. If action C was removed from the net system in Fig. 17a then $>_1 = \{(\mathsf{A}, \mathsf{B}), (\mathsf{B}, \mathsf{D})\}$. With regard to increasing k, it becomes clear that the effect of skipped actions becomes less.

Optional actions are present in some traces of a system, but can be skipped in others. In successor relations, optionality does not remove relation pairs, but adds more, because they describe the potential execution order. In Fig. 17b, actions B and C are optional, whereas skipping B implies skipping C. This adds transition pairs (A, D) and (B, D) to $>_1$, cf. Tab. 1a and Tab. 1b. Again, increasing k reduces the sensitivity of the up-to-k-successor relation, i. e., $>_3$ is equal in Tab. 1a and Tab. 1b.

Concurrency

Concurrency, cf. [72, 189], allows executing several actions in parallel. If two or more actions are enabled concurrently, cf. Def. 3.12, trace semantics treat them in interleaving order [125]. With many concurrent actions, this leads to an exponential growth of the state space and of the number of potential traces of a system [280]. Successor relations abstract from this issue in that concurrent actions are represented by symmetric relation pairs. Figure 17c shows a net system, where B and C are enabled concurrently and hence $(\mathsf{B}, \mathsf{C}), (\mathsf{C}, \mathsf{B}) \in >_k$ for $k \geq 1$, cf. Tab. 1c. That is, each action which is concurrently enabled with another action can be executed directly before or after the latter. By this property, behavioral relations remain compact and it becomes more efficient to find commonalities among the execution order of actions. For instance, the net system of Fig. 17c is able to produce the sole trace of the sequential system in Fig. 17a, which yields that all up-to-k-successor relations of Tab. 1a are contained in Tab. 1c.

Iteration

Cyclic structures, cf. [72, 161], also entail an interleaved ordering of actions, as iterations repeat subsequences of a trace. In Fig. 17d, B and C are in a cyclic structure, i. e., B can be executed before and

after C, which results in a symmetric relation between these transitions, similar to concurrency. While this aspect can be considered a desirable commonality, because both actions are executed in any order, cyclic structures can be distinguished by the parameter k. If the length of subsequences of a trace induced by k is equal or greater than the number of actions in a loop, or in particular if $k \geq b(S)$, and a transition t is within a loop, it holds that $(t, t) \in >_k^S$, because it is possible to execute an action after itself eventually. This is not the case for concurrency, which is illustrated by the self-relation of actions B and C of $>_2$ in Tab. 1d. Successor relations only capture potential execution order of actions, and therefore, cannot capture the cardinality of iterations.

An issue arises if a loop incorporates large parts of or even the complete net system, i. e., a short-circuit net, cf. Def. 3.5. In that case, $>_{b(S)}^S$ supersedes any other relationship among transitions of the net system, as any action can be executed as a successor to any other action. In practice such loops are uncommon [157]. Nevertheless, relations with k smaller than the length of subsequences repeated through iterations are not affected by this drawback and can still identify commonalities and differences of the remaining process parts. If $k = 1$ only self-iterations of single actions are captured.

Choice

A choice between actions, cf. [72, 161], for instance, actions B and C in Fig. 17e, is captured by exclusiveness of the actions in the successor relations, e. g., $(\mathsf{B}, \mathsf{C}), (\mathsf{C}, \mathsf{B}) \notin >_k$. One can conclude that a pair of actions cannot be executed directly after one another if it is not in $>_1^S$, whereas the absence of a pair of actions in $>_{b(S)}^S$ denotes that these actions are mutually exclusive in any trace of the process, as for example in Tab. 1e. This property can be used to identify commonalities in the mutual exclusion of actions in different process models.

If the choice is part of a loop, alternative actions may not be exclusive anymore and may become successors. Therefore, it is reasonable to acknowledge them as a commonality with a process that allows executing these actions in a sequence, e. g., by iteration or concurrency. Moreover, unless an action can appear twice in a trace, e. g., it is in a loop or can be concurrently self-enabled [312], it is exclusive to itself.

Start of a process

The start of a process has been discussed as a differentiating aspect for the behavior of process models and concerns the actions that can be executed initially in a business process [72, 67]. The successor relations assumed as the basis for this work provide only limited support to identify the start of a trace, i. e., point out which actions can be initially executed in a process. Clearly, any action that is contained in behavioral relations as a predecessor, but has no predecessor itself, can be identified as an initial action. However, from the above discussion on optionality and interleaving order of concurrency follows that also actions that are initial actions, may occur as successors in relation pairs. This holds analogously for terminating actions. From

our point of view, this imposes no restriction, as we are interested in commonalities in the order of actions. If a process offers additional actions at its start or end, this does not affect any remaining commonalities. A specific successor relation that solves this issue and is capable of identifying initial and terminating actions is presented in [314].

In [72, 189], also conditions for the execution of actions and dependencies between them are considered. These are abstracted from, as successor relations capture only the potential execution order of actions and not causality. As a solution, the co-occurrence relation has been proposed [318], which expresses that the occurrence of one action implies the occurrence of another. Moreover, from properties of distinguished net classes certain implications can be drawn, e. g., the up-to-k-successor relation with the successor bound, $k \geq b(S)$, is transitive for sound, free-choice workflow systems [314], which allows reasoning on dependencies between actions.

Suitability of Behavioral Relations for Search

Section 4.1 presented several notions to capture and compare the behavior of business processes, including process equivalences. From the existing work on comparing the behavior of process models, the broad majority resorted to interleaving semantics, comparing process models by the traces they can produce. More precise semantics, e. g., bisimulation, are discouraged, as they are too strict in accepting commonalities among business processes.

It has been argued that process models, in particular if they are used to communicate knowledge rather than for implementation, may be restricted in their scope to the positive flow, a simplification of the underlying process, also referred to as "happy path" [4, 208, 263]. Such models typically abstract from iterations and exception handling, which are only introduced at certain levels of detail [4, 208]. While trace equivalence is already considered the weakest notion among the spectrum of behavioral equivalences [301], it discriminates processes where the order of actions is changed or actions are skipped. This, in turn, negatively affects comparative approaches that are based on equivalences of subsequences. Nevertheless, process model search should be able to discover all related models.

Successor relations provide an abstraction over trace semantics and comparing these relations is less sensitive to differences in the traces a process model can produce. From the above discussion on identifying behavioral commonalities, we argue that successor relations are an adequate means that focuses on the execution order of actions, while optionality, causal dependencies, and iteration cardinalities are negligible. For instance, successor relations of a query are mirrored in the successor relations of a match, even if the match has more

complex behavior. As such, optionality is superseded by sequential order, which is in turn superseded by interleaving order of actions.

With regard to the applicability of the presented concepts, we observe that the major limitation is the requirement of boundedness of process models. That is, process models that cannot be traced back to bounded net systems, i. e., models that have an infinitely large state space, are not considered by our approach. For unbounded net systems the computation of successor relations may not be decidable. However, unboundedness is commonly seen as a behavioral error of a process model [282] that should be resolved prior to making the model available for reuse in a process model repository.

In [313], it has been shown that successor relations provide a good approximation of the behavioral consistency between business process models. While consistency focuses on the definite preservation of behavioral aspects, and therefore emphasizes the identification of differences, we argue that search focuses on the identification of commonalities and, for that reason, should have a less discriminating character.

4.3 SEARCHING PROCESS MODELS BY BEHAVIORAL DISTANCES

Following the identification of behavioral commonalities of business processes by the help of their corresponding successor relations, this section elaborates on how these commonalities can be quantified, to evaluate how much a process complies with the behavior of another one. Therefore, we introduce the concept of a *behavioral distance*.

Distances allow not only for the quantification of commonalities in the behavior of two process models, but they also enable the comparison of more than two process models with a reference process model. That is, if distances from each model in a set to the reference model are computed, one can determine which one is closer or more distant to the reference than another. This is the essential property required to search for process models and rank them with regard to a query.

In his influential work on similarity, Lin [171] suggests to measure the distance of two objects by a description of these objects in comparison to the description of their commonalities. Distance and similarity are concepts closely related; as Tversky suggests, they are indeed complementary [274]. We adopt the notion of distance here, as it provides some desirable properties without interfering with established terms such as similarity. Consequently, behavioral distance is a function of common features and distinctive features of the behavioral relations of a pair of process models. We define the distance between behavioral relations in an abstract way that captures future incarnations of that distance, for instance, by successor relations.

Definition 4.1 (Behavioral Relation Distance)**.**
Let $S = (N, M_0)$ and $S' = (N', M'_0)$ be net systems with $N = (P, T, F)$ and $N' = (P', T', F')$. Let $R \subseteq T \times T$ and $R' \subseteq T' \times T'$ be behavioral relations of net systems S and S', respectively. The universe of behavioral relations is denoted by $\mathcal{R}$, and $R, R' \in \mathcal{R}$.

A *behavioral relation distance* is a function $d : \mathcal{R} \times \mathcal{R} \mapsto \mathbb{R}$ that quantifies the commonalities of the behavioral relations, such that if all behavioral aspects of interest of R' are common to R, then $d(R', R) = 0$. ◀

Abstract distance

By R' and R we refer to abstract placeholders for the behavioral relations of the query and candidate model, respectively. Note that it is not required that R and R' are of the same type of behavioral relation. The distance function d is an abstract function that captures the essence of any behavioral distance computed from R' and R, i. e., it denotes a maximum degree of commonality by $d(R', R) = 0$. This allows for definitions of d that are not normalized. Edit distances, cf. [96], that count the number of operations to transform a graph (or relation) into another one are an example for such distance functions.

A concrete instantiation of a distance function requires also concrete instantiations of the behavioral relation types R' and R. Examples are given in Sect. 3.3, e. g., the precedence relation $>_1$. Moreover, these relations can be instantiated by other definitions, cf. [314], or even a combination thereof. The concrete distance function then compares the behavioral relations and provides a distance value from identified commonalties and differences. Independent from their instantiation, distance functions based on behavioral relations are consistent with regard to certain properties.

Properties of Behavioral Distance

Equivalence classes

Behavioral relations are an abstraction of the traces of process models, cf. Sect. 4.2, and hence, models with different sets of traces may have identical behavioral relations.

Definition 4.2 (Equivalence of Behavioral Abstractions)**.**
Let $\mathcal{S}$ be a set of net systems and $\mathcal{R}$ a set of behavioral relations, respectively.

Behavioral abstraction is a function $a : \mathcal{S} \mapsto \mathcal{R}$ that maps a net system S to a behavioral relation R, and defines a partition $\pi(\mathcal{S}) = \{\pi_1, \pi_2, ...\}$ of pairwise disjoint subsets of $\mathcal{S}$ whose union is $\mathcal{S}$, such that for all $S, S' \in \pi_k$ holds that $a(S) = a(S')$.

$\pi(\mathcal{S})$ induces an equivalence relation over pairs of elements of $\mathcal{S}$ that satisfies the properties *reflexivity*, *symmetry*, and *transitivity*, such that for all $S, S' \in \mathcal{S}$ holds

- $S = S' \implies a(S) = a(S')$
- $a(S) \neq a(S') \implies S \neq S'$ ◀

Behavioral relations construct equivalence classes over net systems, such that if two Petri net systems show equivalent behavior, e. g., trace equivalence or bisimulation equivalence, they will also have equivalent behavioral relations. Conversely, if two systems have different behavioral relations, they cannot be trace equivalent or bisimulation equivalent. This has been proven in [314] along with more detailed discussions on equivalences of behavioral relations.

For an example, reconsider the net systems in Fig. 17 with their behavioral relations visualized in Tab. 1. Here, $>_3$ of Tab. 1a and Tab. 1b are equal despite their different sets of traces.

We briefly examine properties of distance functions that are consistent with our framework.

NON-NEGATIVITY, SCALE Intuitively, any distance cannot be negative, i. e., $d(R', R) \geq 0$, as $d(R', R) = 0$ indicates the maximum degree of commonality, cf. Def. 4.1. Distances are, in general, not upper bounded. If $d : \mathcal{R} \times \mathcal{R} \mapsto [0, 1]$, we call the distance function normalized.

REFLEXIVITY, IDENTITY Following from the requirements of Def. 4.1, a relation has all aspects of interest in common with itself and therefore $d(R, R) = 0$. Following from Def. 4.2, the inverse does not hold, i. e., $d(R', R) = 0 \implies R' = R$, a distance value of 0 does not imply identical relations.

SYMMETRY $d(R', R) = d(R, R')$ denotes symmetry of a distance function. In general, the distance function is not required to be symmetric, as the judgement of a distance is associated with the choice of the reference [274]. That is, if the distinctive features of one relation are considered more important than those of the other, the distance function d is asymmetric. If this is not the case, then d is symmetric, and we call it a *similarity measure*.

MONOTONICITY The distance function must be monotonic with regard to common and distinctive features of compared behavioral relations [274]. Monotonicity provides a meaningful ordering of several relations with respect to a common reference. Hence, the distance should decrease when common features are added or distinctive features are removed. Inversely, with the addition of distinctive features to either model the distance should increase. If d is not symmetric, d may not change, when distinctive features are added or removed from a model whose distinctive features are considered less important.

Computation of Behavioral Relation Distances

An instantiation of the abstract distance function introduced above requires concrete instantiations of the behavioral relations R' and R, and a means to compare the relation pairs of R' with those of R, as

Process alignment

the net systems they are based upon comprise distinct sets of transitions, cf. Def. 4.1. In order to compare relationships between actions of two process models, the concept of process alignments has been introduced in Sect. 3.4. Process alignments have been inspired by the alignment of conceptual models in the area of schema and ontology matching [242, 50, 316]. An alignment $(\backsim, \phi)$ captures sets of corresponding transitions of two net systems, i. e., $\backsim \subseteq T' \times T$, and a confidence value between 0 and 1 of the quality of the alignment, $\phi : \backsim \mapsto \mathbb{R}$, cf. Def. 3.13. A pair of transitions is considered a correspondence, if these transitions represent semantically related functionality in both process models. By such an alignment, we can compare the behavioral relations of one net system with those of another net system. We recall that an alignment does not necessarily cover all actions of a process model and, by that, transitions of a net system. In either process, there may exist actions that find no counterpart in the other. As a result, a pair of process models can only share behavioral relations over the set of aligned actions.

Common behavioral relations

Typically, commonalities among relations are computed by the intersection set. However, as behavioral relations of two distinct process models have no common elements, we define the intersection based on the alignment.

Definition 4.3 (Intersection of Behavioral Relations)**.**
Let R and R' be behavioral relations of two net systems $S = (N, M_0)$ and $S' = (N', M_0')$, with $N = (P, T, F)$, $N' = (P', T', F')$ such that $R \subseteq T \times T$, $R' \subseteq T' \times T'$, and $(\backsim, \phi)$ an elementary alignment of their sets of transitions, i. e., $\backsim \subseteq T' \times T$.

The *intersection of behavioral relations*, denoted by $(R' \mathbin{\tilde{\cap}} R) \subseteq \backsim \times \backsim$, consists of corresponding transition pairs, such that for each pair of correspondences $\backsim_x = (x', x)$ and $\backsim_y = (y', y)$ holds that

$$(\backsim_x, \backsim_y) \in (R' \mathbin{\tilde{\cap}} R) \iff (x', y') \in R' \wedge (x, y) \in R \qquad \blacktriangleleft$$

As a result from the above definition, the intersection of behavioral relations is neither a subset of R' nor of R, as it consists of pairs of correspondences. Relation pairs of either behavioral relation that are accounted for by the correspondences can be easily derived, nonetheless. By "shared" or "common" relations of two models, we refer to the intersection of behavioral relations induced by an alignment, hereafter.

The cardinality of correspondences of an alignment in Def. 3.13 deserves discussion, because it is possible that a transition of one net system corresponds to more than one transition of a second net system. Such complex alignments allow for the compensation of different levels of granularities in the examined process models. Following Def. 4.3, this may lead to properties that are counterintuitive, e. g., the intersection of behavioral relations may have more elements than one of the relations. Such anomalies require dedicated exam-

ination. Moreover, the semantics of a set of actions in one model that correspond with a single action in another model needed to be incorporated. If, for instance, the actions in the set are mutually exclusive, this needs to be distinguished from them being executed in a sequence, and accounted for in comparing the behavioral relations, respectively.

Due to the prevailing challenges of automatically identifying complex correspondences [316], we resort to elementary alignments of process models and refer to complex alignments in future work. This is in line with related work toward process model search that is limited to alignments, where one transition is aligned with at most one other transition. Such elementary alignments implicitly assume that actions are modeled on a comparable level of abstraction, and any difference in the modeling granularity affects the identification of commonalities in a negative manner; yet they do not disturb it altogether. In many cases, the assumption that operational process models are captured on a comparable level of granularity is reasonable, cf. Sect. 2.1.

Measures of commonalities

So far, we have introduced the technical grounding to formulate concrete measures of behavioral commonalities, i. e., distances, to quantify behavioral commonalities of process models. De Medeiros et al. [66] leverage behavioral precision and recall as measures based on the set of observation equivalent traces. Precision and recall, originally proposed as quality measures in information retrieval [304], evaluate the fraction of commonalities of an object shared with another one. These measures bear interesting semantics in the context of search. Assume that we have a query model and one of potentially many candidate models with their behavioral relations R' and R, respectively. With the help of the intersection introduced in Def. 4.3, we define precision and recall for behavioral relations.

Definition 4.4 (Precision & Recall of Behavioral Relations)**.**
Let R and R' be behavioral relations of two net systems $S = (N, M_0)$ and $S' = (N', M_0')$, with $N = (P, T, F)$, $N' = (P', T', F')$ such that $R \subseteq T \times T$, $R' \subseteq T' \times T'$, and $(\phi, \backsim)$ an elementary alignment with $\backsim \subseteq T' \times T$.
Functions precision $p : (\mathcal{R} \times \mathcal{R}) \mapsto \mathbb{R}$, and recall $r : (\mathcal{R} \times \mathcal{R}) \mapsto \mathbb{R}$, quantify commonalities between R' and R by a value between 0 and 1 and are defined as follows.

$$p(R', R) = \frac{|R' \,\widehat{\cap}\, R|}{|R|} \qquad r(R', R) = \frac{|R' \,\widehat{\cap}\, R|}{|R'|}$$ ◀

Recall

In the case of search, precision and recall can be used to evaluate how well a candidate matches with the query. *Recall* evaluates the amount of common relation pairs against the relation size of the query, i. e.,

provides a measure of completeness. If each relation pair of the query can be aligned with a relation pair in the candidate, it has a recall value of 1; a value of 0 indicates that none of the query relation pairs can be matched. Any value in-between suggests a partial match, i. e., some but not all relation pairs required by a query were contained in the candidate. In contrast, *precision* captures the fraction of common behavioral relation pairs by the number of relation pairs in the candidate model, or how much behavior of the candidate has been required by the query. A precision value of 0 indicates that query and candidate share no behavioral relation pairs. A value of 1 declares that the candidate has no additional behavioral relation pairs over the relation shared with the query. However, the candidate model may have less relation pairs than the query model, which will be denoted by a recall value below 1. The more additional behavioral relation pairs a candidate exposes, the smaller its precision value will be. This is useful to compare several candidates with equal recall values: The higher the precision of one model, the closer it is, informally speaking, to the query.

Precision

Corresponding distance functions for precision and recall of behavioral relations comply with the properties stipulated in Sect. 4.3. From their definition, it is clear that they are non-negative, normalized, and reflexive. Neither distance function is symmetric, and monotonicity is only satisfied with regard to one relation, i. e., precision is insensitive to the distinctive features of R' and recall to the distinctive features of R.

For an example, recall the order handling query and candidate process models introduced in Sect. 4.2. We refer to the net system of the query in Fig. 16a as S' and of the candidate in Fig. 16b as S. As both net systems are sound, free-choice workflow systems, their behavioral relations can be computed efficiently from their structure [319]. Here, we compare their up-to-$b(S)$-successor relations, shown in Tab. 2. $>^{S'}_{b(S')}$ denotes the relation of S' and $>^{S}_{b(S)}$ of S, re-

	A′	B′	C′	D′	E′
A′	2	1	1	1	2
B′	1	2	2	2	3
C′	·	·	·	1	1
D′	·	·	1	·	1
E′	·	·	1	·	·

(a) $>^{S'}_{k}$ of query net system in Fig. 16a.

	A	C	D	E	F	G
A	·	1	1	2	3	1
C	·	·	1	1	1	·
D	·	1	·	1	1	·
E	·	·	·	·	1	·
F	·	·	·	1	·	·
G	·	·	·	·	·	·

(b) $>^{S}_{k}$ of candidate net system in Fig. 16b.

Table 2: Up-to-k-successor relations for exemplary order processes of Fig. 16. Highlighted fields indicate the intersection of behavioral relations $>^{S'}_{b(S')} \overset{\frown}{\cap} >^{S}_{b(S)}$.

spectively. The correspondence relation of the alignment $(\backsim, \phi)$ of these processes is declared by equal letters of the transition labels, e. g., $(\mathsf{A}', \mathsf{A}) \in \backsim$. Transition pairs of $>^{S'}_{b(S')}$ and $>^{S}_{b(S)}$ that are involved in the intersection of the behavioral relations are highlighted in the relation matrices in Tab. 2, from which results that $|>^{S'}_{b(S')} \hat{\cap} >^{S}_{b(S)}| = 7$. The candidate shows a moderate recall, $r(>^{S'}_{b(S')}, >^{S}_{b(S)}) = 7/15 \approx 0.47$, which indicates that less than half of the behavior requested by S' is provided in S. This is mainly due to a missing correspondence of action B' in the candidate process; since B' can be involved in any process instance, it has weak order relationships with any other action.

This peculiarity is mitigated by shorter look-aheads, i. e., smaller k in the up-to-k-successor relation. For instance, $r(>^{S'}_{1}, >^{S}_{1}) = 6/9 \approx 0.67$. A second factor for the low recall is the loop at the beginning of the process, cf. Fig. 16a, which adds $(\mathsf{A}', \mathsf{A}')$, $(\mathsf{B}', \mathsf{B}')$, and $(\mathsf{B}', \mathsf{A}')$ to the behavioral relation. This iterative behavior is not matched by the candidate. The candidate model S shows also additional behavior over S', which is indicated by the precision value $p(>^{S'}_{b(S')}, >^{S}_{b(S)}) = 7/13 \approx 0.53$, caused by the additional actions F and G.

Precision and recall can be combined into an aggregate measure by their harmonic mean, also referred to as the f-score [86]. Since this is normalized, subtracting it from one yields a distance that increases, the less behavioral relation pairs are shared. The resulting f-score distance

f-score distance

$$d_f(R', R) = 1 - 2 \cdot \frac{p(R', R) \cdot r(R', R)}{p(R', R) + r(R', R)}$$

has a maximum value of 1 if both processes show no commonalities in their behavioral relations, and a minimum distance value of 0 if they show maximum precision and recall, i. e., they have identical behavioral relations. This is in line with Def. 4.1. The f-score distance, d_f, is symmetric due to the aggregation of precision and recall, which are inverse functions on R' and R. For the above example, we conclude with $d_f(>^{S'}_{b(S')}, >^{S}_{b(S)}) = 0.5$ and $d_f(>^{S'}_{1}, >^{S}_{1}) = 0.6$.

Searching with Behavioral Relation Distances

Above, we showed how behavioral relation distances can be computed from behavioral relations, which are an abstraction of process models' behavior. Conceptually, we lift these techniques to process model search by comparing a given query against each candidate that is stored in a process repository, referred to as exhaustive or *sequential search*. This is visualized in Fig. 14, by the cascading rectangles on the right which represent a set of candidate models. Search requires a concrete instantiation of a distance function that reflects the assumptions and expectations of humans that use the search.

Sequential search is comparatively expensive, as every model of the repository is considered for comparison. The actual comparison of behavior is conducted only on the behavioral relations of the can-

Sequential search

didates, which can be precomputed whenever a model in the process repository is stored or updated. Hence, the actual computation time for behavioral relations is of minor significance. Moreover, there exist techniques to reduce the number of comparisons required. They are, however, bound to specific search techniques and will be discussed in Chap. 5 and Chap. 6. We formulate the search problem by means of behavioral distances and the corresponding search result as follows.

Definition 4.5 (Search, Search Result)**.**
Let $R' \in \mathcal{R}'$ be the behavioral relation of a query, $\mathcal{R}$ a set of behavioral relations of the same type, and $d : \mathcal{R}' \times \mathcal{R} \mapsto \mathbb{R}$ a distance function.
Search is a function $\Omega : \mathcal{R}' \mapsto \mathcal{R}^*$ that assigns to a query a sequence of matches.
For a *search result* $\Omega(R') = \langle R_1, R_2, ..., R_n \rangle$, $n \in \mathbb{N}$, holds

$$\forall\, i, j \in \mathbb{N}, 1 \leq i, j < n, i \neq j : R_i \neq R_j$$
$$\forall\, i \in \mathbb{N}, 1 \leq i < n : d(R', R_i) \leq d(R', R_{i+1})$$ ◂

Without loss of generality, we refer to a match in the result by an increasing index of the match's position in the sequence. The conditions stated in Def. 4.5 require that each match is contained at most once and that the search result is ordered by increasing distance to the query, as provided by distance function d. The latter property provides the concept of a ranking by the distance function and requires d to be monotonic to bear a useful meaning. A search result needs to be ranked to present the most relevant matches first to the user, as it has been stated as a requirement for process model search in Sect. 2.4. As each match, and possibly also each candidate, is evaluated against the query, it is imperative that the behavioral relations of the candidates are of the same type that is accepted by the distance function. If it is clear from the context or if the query is of minor relevance, we refer to a search result by Ω, hereafter.

A search result may comprise all process models in the repository, that is, all candidates are also matches. In practice this is of limited use, since candidates with a large distance are no meaningful matches as they share few or no commonalities with the query. Therefore, the number of matches is typically limited. Two fundamental alternatives are distinguished: range queries and nearest neighbor queries. While these terms have their origin in similarity search [333], they apply to search with distances in general.

Range query

Range queries specify a threshold that denotes the maximum distance that a match may have to the query with respect to the distance function used for search — its *range*. The size of the search result is then defined by the number of candidates that satisfy the threshold. A natural threshold would be a distance supremum indicating that no commonalities between query and candidate have been discovered. In the above example, this would require that for each match

$R_i \in \Omega$ it must hold that $d_f(R', R_i) < 1$. The fixed threshold of a range query has a disadvantage, in that, the number of results cannot be known in advance, and it is difficult for users to predict a meaningful range for some distance function. Moreover, different distance functions are likely to have different scales and, as a consequence, are not comparable by range queries.

Nearest neighbor query

In contrast to determining a threshold, *nearest neighbor queries* explicate the size of the search result, i. e., choose the top n matches from the result sequence, and the maximum accepted distance is denoted by the distance of the n^{th} match to the query. It is also possible to combine range and nearest neighbor, e. g., by accepting up to n matches that have a maximum distance to the query. Moreover, the actual decision which candidate is a match could be implemented in a separate function and the distance is merely used for ranking. Nevertheless, the restriction of the number of matches in a search result yields a maximum distance.

Search radius

We refer to this distance as *search radius*, hereafter. Since its value results from the configuration of the query, either explicitly with a threshold of distance or implicitly by the number of search results, we denote it as a property of the query.

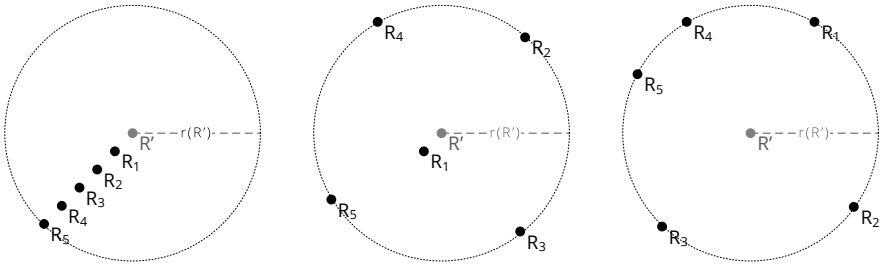
Notation 4.1 (Search Radius). Let $\Omega(R') = \langle R_1, ..., R_n \rangle$, $n \in \mathbb{N}$, be the search result of R'. The *search radius* of R' is denoted by

$$r(R') = \max_{i \in \mathbb{N}, 1 \leq i \leq n} d(R', R_i) \qquad \blacktriangleleft$$

Search result and search radius are concepts defined in the absence of any coordinates, i. e., they are only captured by the distance of matches to the query. For the purpose of discussing various aspects of behavioral distances in search results, we visualize a search result as a sphere centered in the query, hereafter. The radius of the sphere denotes the search radius of the query. This has been inspired by the visualization of metric spaces and distance based search in general, cf. [276, 333], and allows for an appealing way of pointing out subsets of candidates or the search result, e. g., matched against non-matched candidates. A set of different search results is presented in Fig. 18. Note that the planar distribution of search results is arbitrary and bears only limited semantics. In particular, the visualized distance between a pair of search results does not necessarily represent their actual distance induced by the distance function.

4.4 ANALYZING THE QUALITY OF SEARCH RESULTS

To be effective, search shall provide results of high quality, whereas the quality of a search result is typically assessed by the relevance of the provided matches as perceived by users. Deciding whether a result is relevant to a given query is subjective, as it depends on the intention of the user formulating the query and their assumptions



(a) Linearly distributed search result.

(b) Search result with one outstanding match.

(c) Search result with no meaningful ranking.

Figure 18: Illustration of different similarity search results with equal implicit ranking.

about the result [59]. Wang et al. [309] argue that quality attributes with regard to relevance are ideally collected from data consumers. In the same context, the authors propose attributes for *data quality* beyond relevance. These are accessibility, intrinsic quality, and representation. Accessibility denotes that the data should be accessible by users who need them, which is addressed by the management functions of a process model repository, cf. Sect. 2.3.

Data quality

Intrinsic data quality denotes the quality of the underlying data, i. e., the quality of process models, such as readability, coherence, modularity, and the absence of errors [305, 203]. Also reputation is addressed by intrinsic factors, e. g., the creator of the model, the number of successfully terminated process instances, the number of model revisions, and community ratings, cf. [12]. Yet, in many process model repositories, such information is not available.

Finally, representational quality concerns how much information a search result conveys and how easy it is to interpret the results and understand them [309]. This is a crucial premise in order to find the desired match within the search result proposed for a given query. Beyer et al. [34] observed that the meaningfulness of a search result deteriorates in high dimensional spaces. The same issue applies to search based on distances, e. g., between the behavioral relations of process models: If the distance of all matches of a search result to the query is almost identical, the ranking loses its discriminating character and the search result is less likely to be meaningful to the user.

Representation of search results

Figure 18 illustrates three search results obtained with different distance functions, which all produce the same ranking of the search result, $\Omega = \langle R_1, R_2, R_3, R_4, R_5 \rangle$. However, their quality and meaning significantly differs. The first result, cf. Fig. 18a, shows a uniform distribution of distance differences between the matches. As the distances increase linearly, we are able to derive a non-ambiguous ranking for the matches of the search result. In the second result, cf. Fig. 18b, the best result, R_1, is notably better than the other results. In the third search result, cf. Fig. 18c, all results are equally far away from the

query model, i. e., the results are not differentiated in distance. No result is any closer to the query than another. Hence, the ranking is ambiguous and of limited use.

In this section, we propose five evaluation measures that address the representational quality of a search result. In absence of a judgement on the relevance of matches for each possible query, these measures examine distribution aspects within a search result to evaluate confidence, discrimination, and the ranking implicitly provided by the search result. These measures assist a searcher in disseminating a search result before exhaustively inspecting returned models, and hence, facilitate reuse of the returned matches.

Characteristic Distances

The evaluation measures to assess the structure and arrangement of a search result rely only on the matches contained in Ω and the distances between them and the query, that is, we do not assume any particular knowledge about the query, nor do we compare the matched models with models that are not included in the search result. These measures are further independent of the search technique, i. e., they apply to any approach toward search that uses a distance function to produce the ranking of a search result. However, we will reuse the notation for behavioral relations introduced above to remain consistent with the rest of this thesis. This setting allows leveraging evaluation measures as a post-processing step after the computation of the search results.

For the proposed measures, we rely on three characteristic distance functions of the result sequence.

Notation 4.2 (Characteristic Distances of a Search Result). Let $\Omega = \langle R_1, R_2, ..., R_n \rangle$ be a search result of a query $R' \in \mathcal{R}'$ over a collection of process models $R_i \in \mathcal{R}$, and $d : \mathcal{R}' \times \mathcal{R} \mapsto \mathbb{R}$ a distance function according to Def. 4.1.

- $min(\Omega) = d(R', R_1)$ is the minimum distance, i. e., the distance of the element in the search result closest to the query.
- $med(\Omega) = d(R', R_i), i = \lceil \frac{n}{2} \rceil$ is the median distance to the query, which intuitively resembles the maximum distance of the "better half" of the results. We also refer to the better half of results as *most results*, hereafter.
- $max(\Omega) = d(R', R_n)$ is the maximum distance, i. e., the distance of the element in the search result farthest from the query. ◂

All evaluation measures presented hereafter express search result statistics and produce values $x \in \mathbb{R} \wedge x \in [0, 1]$, where the best case is expressed by a value of 1 and the worst or sufficiently unsatisfying cases by a value of 0.

Result Confidence

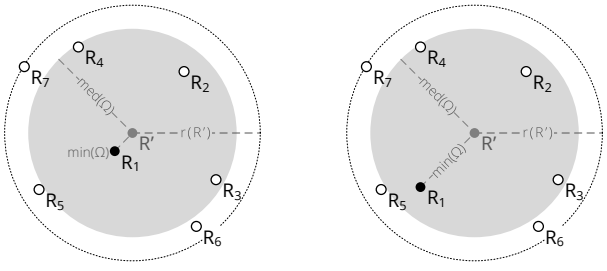
Distinction of search results

Confidence expresses the belief that the best results are well distinguished from the perspective of the rest of the results. This is expressed by the distance to a reference value that determines the maximum distance of the results to be compared. In order to assess the confidence of the best result, i. e., the closest match, we compare its distance to the query model with the median distance of the result sequence. The median is more stable and not as strongly affected by the number of results retrieved as the arithmetic mean or maximum distance in the search result because the median is generally robust against outliers in a data set. Figure 19 shows two examples with opposing quality, where the gray shaded circle demarcates $med(\Omega)$, i. e., the superior search results are contained within the circle, the inferior are excluded.

Definition 4.6 (Confidence best match)**.**
Let Ω be search result. The *confidence of the best match* is a function $\mathcal{C}_1 : \Omega \mapsto [0, 1]$ such that

$$\mathcal{C}_1(\Omega) = \begin{cases} 1 - \frac{min(\Omega)}{med(\Omega)} & \text{if } med(\Omega) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$ ◀

In the best case, the minimum distance to the query in the results equals 0, which implies that according to the applied similarity measure there is at least one model in the search result, which is equal to the query model. A rather high confidence value is illustrated in Fig. 19a, where R_1 (black data point) clearly stands out against the remaining matches (white data points), indicated by a large difference to the median. In contrast, R_1 is not much closer to the query than the rest of the matches in Fig. 19b, and therefore, yields a lower confidence about its ranking as the best match. In the worst case, the minimum equals the median or both equal 0, which implies that the best match is not any better than the majority of the search result.



(a) High confidence best match.

(b) Low confidence best match.

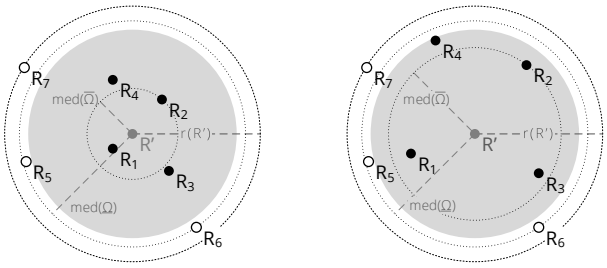
Figure 19: Illustration of result confidence of the best match.

As a confidence measure for the better half of matches, we compare the sequence of superior models of the result, denoted by $\overline{\Omega}$, with the inferior models, $\underline{\Omega}$, dividing the result into two groups.

Definition 4.7 (Confidence most matches)**.**
Let $\Omega = \langle R_1, R_2, ..., R_n \rangle$ be a search result and index $i = \lceil \frac{n}{2} \rceil$.
We define two subsequences $\overline{\Omega} = \langle R_1, ..., R_i \rangle$ and $\underline{\Omega} = \langle R_{i+1}, ..., R_n \rangle$ which partition the result sequence in two sequences.
The *confidence of most matches* is a function $\mathcal{C}_{most} : \Omega \mapsto [0, 1]$ such that

$$\mathcal{C}_{most}(\Omega) = \begin{cases} 1 - \frac{med(\overline{\Omega})}{med(\underline{\Omega})} & \text{if } med(\underline{\Omega}) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$ ◂

A large difference in values $med(\overline{\Omega})$ and $med(\underline{\Omega})$ indicates that the majority of the superior models is significantly better than the inferior models. Figure 20a illustrates such a setting, determined by $med(\overline{\Omega})$ (black data points) and $med(\underline{\Omega})$ (white data points). Contrary, Fig. 20b shows the same partition of the search result into superior and inferior models, yet the superior models are not anymore outstanding against the inferior models, demonstrated by the low contrast between $med(\overline{\Omega})$ and $med(\underline{\Omega})$. Consequently, one cannot reasonably argue that the superior matches are truly better, as they become rather indistinguishable.



(a) High confidence most matches.

(b) Low confidence most matches.

Figure 20: Illustration of result confidence of the superior partition of a search result.

Result Discrimination

Discrimination describes the distribution of the distances between the query and the matches of the search result. Well differentiated matches have a higher probability to show an unambiguous ranking compared to a search result where most of the matches do not differ in their distance to the query model.

Stability of search result

Beyer et al. [34] discussed the meaningfulness of similarity search in high-dimensional vector spaces. They proposed a stability measure based on the capability of a query to obtain matches that can be dis-

tinguished in an unambiguous fashion. Consequently, this provides a measure how well a search result can be discriminated. Whereas the measure proposed in [34] provides only a binary value, i. e., a query either produced a stable result or not, we adapted this notion and extended it, such that it provides a continuous measure.

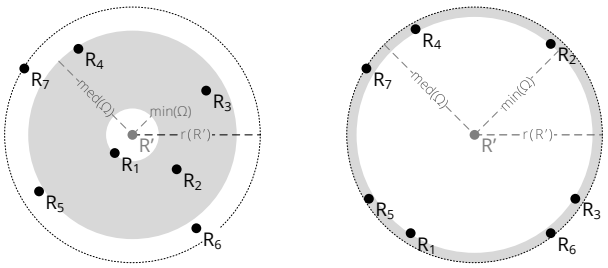
Consequently, the stability of a search result is defined by the interval between the distance of the best match and the median of all matches, normalized by the range of match distances to the query in the search result. Put differently, we compare the range of the superior half of the search result against the whole range of matches.

Definition 4.8 (Discrimination most matches)**.**
Let Ω be a search result. The *discrimination of most matches* is a function $\mathcal{D}_{most} : \Omega \mapsto [0, 1]$ such that

$$\mathcal{D}_{most}(\Omega) = \begin{cases} \frac{med(\Omega) - min(\Omega)}{max(\Omega) - min(\Omega)} & \text{if } max(\Omega) \neq min(\Omega) \\ 0 & \text{otherwise} \end{cases}$$ ◀

Figure 21 depicts two results for the quality measure discrimination of most results. Here, the interval of the range of the superior model is shaded gray. In the worst case, the interval in which most matches lie is 0. A comparable case is shown in Fig. 21b, where the interval is very small, indicated by a narrow range. Most matches are therefore not differentiated in distance to the query model, and a ranking of this search result cannot be unambiguous and is, therefore, considered of limited use. For the other end of the spectrum, Fig. 21a depicts a rather wide interval, illustrated by the width of the ring that contains the superior matches. In the best case, the width of the ring spreads the whole range, which indicates that the majority of matches are well distinguishable.

As $\mathcal{D}_{most}$ does not address the ranking of the inferior search result, we also propose a measure that evaluates the distribution of all matches within the search result. Here, we consider a linear increase



(a) Good Discrimination most matches.

(b) Poor discrimination most matches.

Figure 21: Illustration of result discrimination of the superior partition of a search result.

in distances to the query as the ideal case, i. e., the distance differences between consecutive matches in the search result are equal, which is depicted in Fig. 22a.

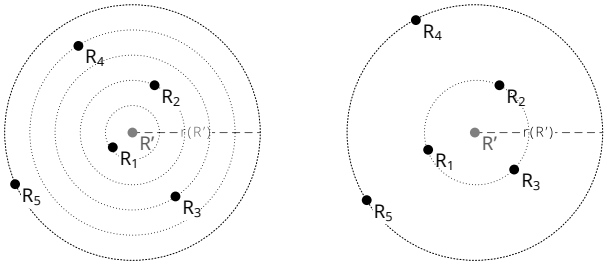
Definition 4.9 (Discrimination all matches)**.**
Let $\Omega = \langle R_1, ..., R_n \rangle$, $n \in \mathbb{N}$, be a search result. For a pair $R_i, R_j \in \Omega$, the inter-result difference function, $ird : (\mathcal{R} \times \mathcal{R}) \to \mathbb{R}$, is defined as follows

$$ird(R_i, R_j) = |d(R', R_i) - d(R', R_j)|$$

The *discrimination of all matches* is a function $\mathcal{D}_{all} : \Omega \mapsto [0, 1]$ such that

$$\mathcal{D}_{all}(\Omega) = \begin{cases} 1 - \frac{\sum_{1 \leq i < n} \left| ird(R_i, R_{i+1}) - \frac{max(\Omega) - min(\Omega)}{n-1} \right|}{2 \cdot (max(\Omega) - min(\Omega))} & \text{if } max(\Omega) \neq min(\Omega) \\ 0 & \text{else} \end{cases}$$ ◀

Here, we compare the inter-result difference of each neighboring pair in a sequence, $\text{ird}(R_i, R_{i+1})$ against the ideal difference provided by the range of the result sequence and the number of inter-result distances, which is one less than the cardinality of the search result, $n = |\Omega|$. Figure 22b shows a scenario with low discrimination of matches, where all matches have either distance $min(\Omega)$ or $max(\Omega)$ and are therefore not well distinguishable. In the worst case, all matches are equidistant to the query and therefore $\mathcal{D}_{all}(\Omega) = 0$.



(a) Good Discrimination all matches.

(b) Poor discrimination all matches.

Figure 22: Illustration of result discrimination of the complete search result.

Ranking Agreement

A vast number of process distance measures, including similarity measures, exists that address linguistic, structural, and behavioral features of process models [22]. In Sect. 4.1, we discussed approaches that incorporate the behavior of processes. With this abundance of measures addressing various aspects of process models, it is reasonable to use one distance function to decide a match and another to rank the matches in the search result. Given one ranking, its agreement with rankings obtained by alternative measures can be assessed.

A high agreement among several rankings suggests that the ranking is consistent with several aspects of process models.

Comparison of different rankings

For this purpose, we propose a measure that incorporates the ranking positions of matches rather than their distance to the query. This is grounded in the possibly different scales of various distance measures, e. g., normalized and non-normalized distances, which would bias a ranking agreement based on distance values significantly. With the ranking agreement we assess to which extent different distance functions agree on the ranking of the matches. Given a search result Ω, we denote a ranking v of the same search results produced by some means by Ω^v. Hence, all rankings over Ω contain the same matches, but they may be ordered differently. A ranking produced by the same similarity function as the one to obtain Ω is called an *implicit ranking*; other rankings could be created by other distance functions, or, for example, through human assisted reordering.

The partial function $rank : \mathcal{R} \times \Omega^v \nmapsto \mathbb{N}$ returns the position of a matching model $R_i \in \Omega$ in a ranking Ω^v. Consider, for example, the following rankings for a search result Ω: $\Omega^1 = \langle A, B, C, D, E \rangle$, $\Omega^2 = \langle C, B, A, D, E \rangle$, and $\Omega^3 = \langle A, C, B, E, D \rangle$. Here, $rank(C, \Omega^1) = 3$, $rank(C, \Omega^2) = 1$ and $rank(C, \Omega^3) = 2$.

Definition 4.10 (Ranking agreement)**.**
Let $\mathcal{M} = \{\Omega^1, ..., \Omega^m\}$, $|\mathcal{M}| = m$, be a set of rankings over the same search result Ω, $|\Omega| = n$, i. e., $\forall 1 \leq u, v \leq m,\ 1 \leq i \leq n$: $R_i \in \Omega^u \iff R_i \in \Omega^v$.

We define the ranking difference function, $diff : \mathcal{M} \mapsto [0, 1]$, as follows:

$$diff(\mathcal{M}) = \sum_{1 \leq u < m} \left(\sum_{u < v \leq m} \left(\sum_{R \in \Omega} |rank(R, \Omega^u) - rank(R, \Omega^v)| \right) \right)$$

The *ranking agreement* is a function $\mathcal{A} : \mathcal{M} \to [0, 1]$ defined as

$$\mathcal{A}(\mathcal{M}) = 1 - \frac{diff(\mathcal{M})}{\binom{m}{2} \cdot \frac{1}{4}(2 \cdot n^2 + (-1)^n - 1)}$$ ◂

For every combination of rankings (Ω^u, Ω^v) and for every match R_i in the search result, we compare the positions of R_i in the respective pair of rankings and sum their absolute ranking difference. In order to normalize the overall ranking difference, we divide by the worst possible ranking difference, determined by the number of ranking combinations, $\binom{m}{2}$, where m is the number of rankings, and the worst possible pairwise ranking difference, which is produced by $\frac{1}{4}\left(2 \cdot n^2 + (-1)^n - 1\right)$. If all rankings agree in the ordering of the matches in Ω, their ranking difference diff$(\mathcal{M})$ will be 0, leading to the best case value of $\mathcal{A}(\mathcal{M}) = 1$.

The above example provides three different rankings of matches A, B, C, D, E of a search result. If we compare the rankings of Ω^1 and Ω^2, we get $diff(\{\Omega^1, \Omega^2\}) = |1 - 3| + |2 - 2| + |3 - 1| + |4 - 4| + |5 - 5| = 4$. The ranking difference of all rankings $\mathcal{M} = \{\Omega^1, \Omega^2, \Omega^3\}$

yields 14, which leads to a ranking agreement $\mathcal{A}(\mathcal{M}) = 1 - 14/36 \approx 0.61$.

In contrast to existing ranking correlation coefficients, for instance, Kendall's τ or Spearman's ρ [57], the ranking agreement $\mathcal{A}(\mathcal{M})$ is sensitive to the ranking distance of a match in several rankings rather than considering concordance or non-concordance[1] only, and it is able to incorporate more than two rankings.

Application of Quality Measures to Process Model Search

While business process model search has become a research topic of considerable interest, techniques are typically restricted in their focus to obtaining and, sometimes, ranking a search result. Questions toward the quality of a search result are generally neglected. As business process model collections usually do not provide meta information which may act as an indicator for the quality of a model, we evaluate search results from a quantitative perspective.

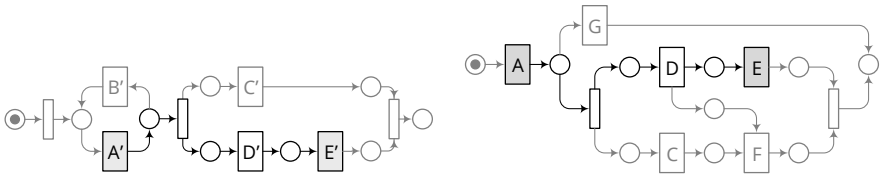
Therefore, we presented measures to judge on the quality of search results, by means of characteristic distances. These measures assess the confidence that the best matches stand out compared to inferior matches, evaluate the ambiguity of a provided ranking, and examine the ranking of a search result with regard to other similarity measures. These measures shall be used to provide an overall evaluation of the search result to the user, to indicate which matches are more likely to satisfy the user's request, or even to improve the ranking of an obtained search result by means of additional distance measures.

The effectiveness of these measures is analyzed in an empirical evaluation in Sect. 7.5 with the help of a business process model collection for which a human assessment of process similarity is available. The evaluation of these measures reflects the tendency of human assessment and can be used to predict the optimal size of the search result by increasing confidence, to provide insights to the user about the discrimination of the ranking, and, given that a ranking is not sufficiently discriminative, can propose other distance functions to improve or support a ranking by means of the ranking agreement. On that account, we argue that our measures give a good indication of the quality of search results and thus, assist the user in assessing the meaningfulness of a provided search result.

4.5 VISUALIZING SUCCESSOR RELATIONS

So far, this chapter has focused on the identification of behavioral commonalities by discovering the intersection of behavioral relations and computing a behavioral distance. Based on such distances, pro-

1 A match in one ranking is concordant with another ranking, if both rankings show the same ranking position for that match.



(a) Query net system. (b) Candidate net system.

Figure 23: Net systems with behavioral common up-to-3-successor relation pairs (A, E) and (A', E') visualized in the net system.

cess model search has been introduced along with quality measures on the background to provide the user with insights into the search result as a whole and to increase its representational quality. In this section, we focus on the representation of a single match and how to support the user in understanding the commonalities identified by comparing behavioral relations, and facilitate reusing models of the search result.

Users that search for process models find it difficult to understand, why a process model matches a query, in particular, if the matching criterion is based on behavior rather than on the process model's graph [66]. Despite the large number of approaches to process model search, visualizing which part of a process model has been matched by a query remains a challenge. On the other hand, it has been shown that highlighting specific parts of a process model supports the comprehension of these parts considerably [165, 245]. Supporting users in understanding matches is crucial for effective search and is, in particular, required if the match is not obvious anymore from carefully looking at the process models.

Applied to the context of process model search by example, successor relation pairs that contribute to the commonalities of a query and a match, cf. Def. 4.3, should be visualized in process models of the search results. An example for visualizing common successor relation pairs is illustrated in Fig. 23. Here, we highlighted successor pairs (A, E) of net system S, cf. Fig. 16b, and (A', E') of net system S', cf. Fig. 16a. Matched transitions are shaded, and paths that represent the behavior comprised by the respective successor relation pair are highlighted bold. Visualization is conducted after deciding a match, i. e., visualization takes a subset of the successor relation of a match, previously discovered as matching the query as input and provides a projection on the net of the match. That is, correspondences need not be resolved and the projection is local to one Petri net.

Approaches to identify common structural aspects of process models, e. g., graph isomorphisms [25, 180] or graph edit distances [169, 73], can resort to subgraphs and paths within the net to visualize structural commonalities that lead to a match. However, matching dynamic, i. e., behavioral aspects, to a static structure is not trivial.

Successor relations are an abstraction of the behavior of process models based on trace semantics, and therefore, a potentially infinite set of firing sequences may exist for each relation pair, cf. Sect. 3.3. Hence, we are interested in the discovery of a finite set of firing sequences that contribute to a successor relation. These firing sequences need to be projected on the net's structure. For the candidate in our example, this means visualizing a subset of firing sequences that lead from A to E.

The projection of firing sequences to paths in a process model or net system, respectively, is not trivial in the general case. For example, not every path in a net leads to a firing sequence. This is depicted in Fig. 24, where t_3 can be executed after t_1 and there exists a path from t_1 to t_3 via t_2. However, t_2 and t_3 are mutually exclusive and highlighting the aforementioned path would suggest an incorrect fragment of the net. The contrary can also be the case, i. e., concurrently enabled transitions can appear in firing sequences after each other although there exists no directed path between them, e. g., the firing sequence $\langle C, D \rangle$ in Fig. 16b can occur due to interleaved transition firing.

Moreover, there may exist firing sequences that have a negative effect on the comprehension of a behavioral relation, i. e., not all firing sequences that yield a successor relation pair should be considered for the projection to the net's graph. In the following, we examine the visualization of behavioral relations more closely and explain which firing sequences we consider to contribute to the user's comprehension. For a general solution, we first focus on bounded net systems, i. e., process models with a finite state space which includes all sound process models. Unboundedness is commonly seen as a behavioral error of process models [282]. Then, we show how this can be realized in a simpler and considerably more efficient fashion for sound, free-choice workflow systems.

Bounded Net Systems

Following Def. 3.10, a pair of transitions that is in a successor relation, $(x, y) \in >_k^S$ of a net system S denotes that there exists a non-empty set of firing sequences that execute these transitions in the given order, i. e., $\sigma = \langle t_1, ..., t_n \rangle$ with $t_1 = x$ and $t_n = y$. These sequences need to be considered for the projection of (x, y) in the net's graph. To visualize such firing sequences, the parameter k of the successor relation is

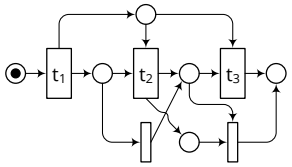


Figure 24: A sound net system that is not free-choice.

irrelevant and, therefore, we resort to the up-to-k-successor relation with successor bound $k = b(S)$ for the remainder of this section.

From these firing sequences, some are undesirable, due to iterations and concurrency, and should be disregarded for visualizations in the net. Iterations result from cyclic dependencies in the net's flow relation, and produce firing sequences that repeat certain subsequences and therefore become infinite. Concurrency concerns the concurrent enabling of transitions and leads to interleaved firing.

Concurrency

Let markings $M_i, M_j \in \mathbb{M}$ enable transition t_i, i. e., $(N, M_i)[t_i\rangle$, and transition t_j, i. e., $(N, M_j)[t_j\rangle$. Then $M_i + M_j$ concurrently enables both transitions. That is, firing either transition does not affect the enabling of the other transition, and firing both transitions in any order directly after each other results in identical markings, cf. Def. 3.2 and Def. 3.3. This is extended to the notion of concurrent firing sequences, where the enabling of any transitions of either sequence is concurrent to, and thereby independent from, the enabling of any transition of the other sequence. The concurrency relation, $||_c^S \subseteq T \times T$, cf. Def. 3.12, captures all pairs of transitions that can be enabled concurrently in a net system. Two firing sequences σ and σ' of a system S are concurrent if for each transition $t \in \sigma$ holds that $\forall\, t' \in \sigma' (t, t') \in ||_c^S$ and vice versa.

Concurrency leads to the following predicament: If two transitions can be enabled concurrently, they will be contained in the successor relation, i. e., $(x, y) \in ||_c^S \implies (x, y), (y, x) \in >_{b(S)}^S$. As a consequence, the projection of a successor relation pair (x, y) that can be concurrently enabled cannot rely on a directed path between the transitions in the net, as concurrency is carried out as firing them in interleaved order. We will postpone this problem for now, require that transitions of a successor relation pair to be projected are connected by a directed path, $(x, y) \in F^+$, and return to it later.

If a transition is concurrent to x or y, it can be executed before x or after y, respectively, and hence, does not add to the firing sequences captured by the behavioral relation $(x, y) \in >_k^S$. Consider the net system in Fig. 23b, where, due to the concurrent enabling of C and E, firing sequences exist that include C as well. However, highlighting a path to C will not increase comprehension of the successorship of A and E.

In the following definition, we establish a notion of minimal firing sequences, which excludes transitions concurrently enabled with the transitions of the successor relation pair to be projected.

Definition 4.11 (Minimal Firing Sequence)**.**
Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$, and $\sigma = \langle t_1, ..., t_n \rangle$ a firing sequence enabled in a reachable marking $M_1 \in [N, M_0\rangle$, i. e., $(N, M_1)[\sigma\rangle$, for which holds $(t_1, t_n) \in F^+$.
For each transition $t_i \in T$ a minimal enabling marking $[t_i] \in \mathbb{M}$, i. e., $(N, [t_i])[t_i\rangle$, denotes that $\forall\, p \in P : [t_i](p) \geq 1 \iff p \in {}^\bullet t_i$.
σ is a *minimal firing sequence* with regard to (t_1, t_n) iff

$$\forall\, 1 \leq i \leq |\sigma| \;\forall\, t_i \in \sigma, (N, M_i)[t_i\rangle(N, M_{i+1}) : M_{i+1} \not\geq M_1 + [t_{i+1}]$$
$$\forall\, 1 \leq i < |\sigma| \;\forall\, t_i \in \sigma, (N, M_i)[t_i\rangle(N, M_{i+1}) : M_i \not\geq [t_i] + M_{|\sigma|} \qquad \blacktriangleleft$$

For a successor pair (x, y) to be projected, the first condition excludes sequences that contain transitions concurrently enabled with x and the second transitions concurrently enabled with y. It follows that any transition of a minimal firing sequence lies on a path from x to y.

Lemma 4.1.
Let $\sigma = \langle t_1, .., t_n \rangle$ be a minimal firing sequence of $(t_1, t_n) \in F^+$. For each $1 < i < |\sigma|$ it holds that $(t_1, t_i) \in F^+$ and $(t_i, t_n) \in F^+$.

Proof. We first show $\forall\, 1 < i \leq n : (t_1, t_i) \in F^+$ by induction over the position of t_i in σ.

(Base) From $\sigma(i) = t_i$ and $\sigma(i+1) = t_{i+1}$ follows that there exists a reachable marking $M_i, M_{i+1} \in (N, M_0)$ before and after firing of transition t_i, i. e., $(N, M_i)[t_i\rangle(N, M_{i+1})$, where t_{i+1} is enabled in M_{i+1}, i. e., $(N, M_{i+1})[t_{i+1}\rangle$. If $(t_i, t_{i+1}) \notin F^+$ then ${t_i}^\bullet \cap {}^\bullet t_{i+1} = \emptyset$, and therefore firing of t_i does not affect the enabling of t_{i+1}. Hence, there must exist a marking $M' \geq M_i + M_{i+1}$. If $i = 1$ this results in a contradiction, as we required that t_1 must not be concurrently enabled with t_2 in Def. 4.11.

(Step) Let $\sigma(1) = t_1$, $\sigma(i) = t_i$, and $\sigma(k) = t_k$ with $1 < i < k$. From $(t_i, t_k) \notin F^+$ follows that there exists a j with $i \leq j < k$ such that $\sigma(j) = t_j$, $\sigma(j+1) = t_{j+1}$ and $(t_j, t_{j+1}) \notin F^+$, i. e., t_j and t_{j+1} are concurrent (see base). In that case, they can be executed in interleaving order. It suffices to show that $(t_1, t_j) \in F^+$, which follows from iterating the step.

The second relation, $\forall\, 1 \leq i < |\sigma| : (t_i, t_n) \in F^+$, follows from the mirrored argument. □

Def. 4.11 and Lem. 4.1 ensure the absence of transitions that are not on a path between the transitions of the successor relation in minimal firing sequences. The step of the proof provides the solution to the aforementioned predicament that concerns the visualization of a behavioral relation pair of concurrent transitions, mentioned above.

In a net system that has a single input place, a pair of transitions (x, y) that are enabled concurrently has always a common ancestor

transition that produces multiple tokens leading to the concurrent enabling and is, therefore, executed before the concurrent transitions. It is straightforward to transform a net system (N, M_0) into another net system that has only one start place by adding the start place and a start transition that puts tokens on all places marked in M_0. From the step of Lem. 4.1, it follows that there exists a path from the common ancestor transition to x and y. In order to visualize the concurrent enabling of these transitions, the nearest common ancestor transition z in the workflow net is discovered and then the minimal firing sequences for $(z, x), (z, y) \in F^+$ are obtained for projection.

Iterations

Cyclic dependencies, i. e., $\exists\, x \in T : (x, x) \in F^+$, in a net system may result in an infinite set of traces, as the cycle can be iterated arbitrarily often, including zero times. Unless either transition of a behavioral relation is part of the cyclic structure, projection of firing sequences that contain the iteration is unlikely to improve comprehension of the behavioral relation. We argue that by projecting insignificant paths the attention to the relevant paths deteriorates.

For example, in a short-circuit net system, cf. Sect. 3.2, the set of sequences captured by a behavioral relation pair covers all firing sequences in the net, as it is possible to iterate over all transitions. Therefore, we argue that sequences that show unnecessary iterations should be excluded. However, sequences that just repeat transition occurrences cannot be disregarded, as this may be due to complex firing conditions. Iterations can only be safely excluded by avoiding that a previously encountered state, with regard to the cyclic structure, is not revisited.

In Petri net systems, a state is represented by a marking M_i, and firing transitions of a sequence $\sigma = \langle t_1, ..., t_n \rangle$ advances the marking of the system, $(N, M_i)[t_i\rangle(N, M_{i+1})$, cf. Def. 3.3. If a marking is visited twice, the sequence contains an iteration. A firing sequence may also contain an iteration without visiting a state twice, if the markings differ only in the enabling of concurrent transitions. This is illustrated in Fig. 25, where transitions t_1 and t_2 are in a cycle. From the depicted marking, the firing sequence $\langle t_1, t_3, t_2, t_1, t_4 \rangle$ would not visit any marking twice, regardless of iterating over t_1 and t_2, due to the concurrent path that comprises transitions t_3 and t_4. Sequences that contain such iterations are excluded by the following definition.

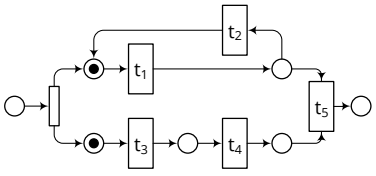


Figure 25: A sound net system with a cycle in a concurrent branch.

Definition 4.12 (Non-Iterating Firing Sequence)**.**
Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$ and $\sigma = \langle t_1, ..., t_n \rangle$ a firing sequence enabled in a reachable marking $M \in [N, M_0\rangle$.

For each transition $t_i \in \sigma$, a transition $t_j \in \sigma$, $t_j \neq t_i$, is enabled concurrently with t_i in σ, iff there exists a subsequence $\sigma' = \langle t_1, ..., t_k \rangle$ of σ with $k \leq n$, such that $(N, M)[\sigma'\rangle(N, M')$ and $M' \geq M_1 + M_2$ with $(N, M_1)[t_i\rangle$, and $(N, M_2)[t_j\rangle$.

Two reachable markings $M, M' \in [N, M_0\rangle$ are different under concurrency, denoted by $M \neq_{||_c} M'$ iff there exist transitions t, t' that cannot be concurrently enabled in σ and $\exists\, p \in (\bullet t \cup \bullet t') : M(p) \neq M'(p)$.

σ is a *non-iterating firing sequence* iff

$$\forall\, 1 \leq i < j \leq |\sigma| \;\forall\, t_i \in \sigma, (N, M_i)[t_i\rangle(N, M_{i+1}) : M_i \neq_{||_c} M_j \qquad \blacktriangleleft$$

The definition requires that for non-iterating firing sequences no state is revisited which equals a previously reached state in the marking of transitions that are not concurrent. With respect to Fig. 25 and upon highlighting $(\mathrm{t}_1, \mathrm{t}_2) \in >^S_{b(S)}$ this excludes the cycle and thereby any sequences that contain transition t_2.

Projecting sequences to paths

In order to project a successor relation to the net's structure, all minimal and non-iterating firing sequences need to be found. This set is finite in bounded net systems and can be computed by reachability analysis, which is decidable for bounded net systems [112]. From the above arguments follows that the set of non-iterating, minimal firing sequences $\sigma = \langle t_1, ..., t_n \rangle$ for a successor relation is represented by a set of paths $(t_1, t_n) \in F^+$. Thus, visualization of a successor relation requires highlighting the paths corresponding to all non-iterating, minimal firing sequences that are captured by the behavioral relation.

From Lem. 4.1 follows that minimal firing sequences $\sigma = \langle t_1, ..., t_n \rangle$ with a pair of direct successors that has no corresponding path, i. e., $(t_i, t_{i+1}) \notin F^+$, are not necessarily excluded. Highlighting a sequence requires iterating over transitions t_1 to t_n. The path $(t_i, t_{i+1}) \in F^+$ can be visualized in a straightforward manner; in case of $(t_i, t_{i+1}) \notin F^+$ visualization is skipped for the pair of direct successors. Since Def. 4.11 only excludes sequences that contain transitions enabled concurrently to transition t_1 and t_n, all paths that correspond to minimal firing sequences will be highlighted. In our example, the set of non-iterating, minimal firing sequences for $(\mathsf{A}, \mathsf{E}) \in >^S_{b(S)}$ is $\{\langle \mathsf{A}, \mathsf{D}, \mathsf{E} \rangle\}$ which leads to the highlighted path illustrated in Fig. 23b.

Sound, Free-choice Workflow Systems

For the distinguished class of sound free-choice workflow systems, the problem of finding all paths in a net that represent a successor relation $(x, y) \in >^S_{b(S)}$ can be solved efficiently, that is, in low polynomial time without an analysis of the state space. Sound, free-choice work-

flow systems, cf. Sect. 3.2, provide several useful properties due to the close coupling of structure and behavior [145]. On the one hand, the up-to-k-successor relation with successor bound $b(S)$ is transitive, i. e., it holds $\forall\,(x,y,z) \in T : (x,y),(y,z) \in >^S_{b(S)} \implies (x,z) \in >^S_{b(S)}$ [314]. On the other hand, successor relations with larger look-ahead subsume those with smaller, i. e., $k' \leq k \implies >^S_{k'} \subseteq >^S_k$ [314], and hence from a set of transitions $\{t_1,...,t_n\}$ for which holds $(t_i,t_{i+1}) \in >^S_1$ follows that $(t_1,t_n) \in >^S_{b(S)}$.

Concurrency

With regard to these properties, projecting a successor relation is straightforward as there exist directed paths between the predecessor and successor transitions in the net that correspond to firing sequences unless the transitions can be concurrently enabled. Hence, it is sufficient to project all pairs of the 1-successor relation of a workflow system that are neither concurrent nor lead to undesirable iterations which have been discussed for bounded net systems.

Lemma 4.2.
Let $S = (N, M_0)$ be a sound, free-choice workflow system. Then, for any pair of transitions in an up-to-k-successor relation, $(x,y) \in >^S_k$, it holds that $(x,y) \in ||^S_c \iff (x,y) \notin F^+$.

Proof.

($\Leftarrow$) From $(x,y) \notin F^+$ follows that the firing of x cannot impact on the enabling of y. From $(x,y) \in >^S_k$ follows that there exists a firing sequence including x and y and neither transition is dead due to the soundness property. Hence, there must exist a marking that enables x and y concurrently, i. e., $M \geq M_x + M_y$ with $(N,M_x)[x\rangle$ and $(N,M_y)[y\rangle$, and therefore $(x,y) \in ||^S_c$.

($\Rightarrow$) By contradiction. From $(x,y) \in ||^S_c$ follows that there exists a reachable marking $M \in [N,M_0\rangle$ that enables x and y concurrently, i. e., $M \geq M_x + M_y$ with $(N,M_x)[x\rangle$ and $(N,M_y)[y\rangle$. Assume there is a path $(x,y) \in F^+$, from which follows due to soundness and free-choice that there exists a firing sequence $\sigma = \langle t_1,...\rangle$, with $t_1 = x$, enabled in M_x and therefore in M, whose firing enables y, cf. [145, Lemma 4.2]. Hence, firing of $(N,M)[\sigma\rangle(N,M')$ adds tokens to input places of y, such that $M' > M_y$ and therefore $\forall\, p \in {\bullet}y : M'(p) > 1$. However, sound free-choice workflow nets are safe [283, Lemma 4], i. e., all places are bounded by 1. This contradicts the assumption. □

Following Lem. 4.2, unless $(x,y) \in >^S_1$ is also in the concurrency relation $||^S_c$, both transitions x, y are connected by a directed path. Concurrent relation pairs can be disregarded for the visualization, as they express interleaved execution order and have no representation as a path in the net, see above.

Iterations

Undesired iterations shall be excluded as well. In [319] it has been shown for sound, free-choice workflow systems that a pair of transitions (x, y) that is in interleaving order, i. e., $(x, y), (y, x) \in >_k^S$, is in a cyclic dependency of the model graph unless the transitions can be enabled concurrently. If a transition appears more than once in a trace, it is in interleaving order with itself. Sound, free-choice workflow systems are safe [283, Lemma 4], i. e., each place has an upper bound of one, and therefore, a transition cannot be concurrently enabled with itself. Consequently, iterations in sound, free-choice workflow nets are avoided by disregarding firing sequences that repeat a transition.

Following from the above arguments, we define the *successor decomposition* for non-concurrent transition pairs in an up-to-k-successor relation, which contains all 1-successors that do not show interleaved execution ordering nor iterations.

Definition 4.13 (Successor Decomposition)**.**
Let $S = (N, M_0)$ be a sound, free-choice workflow system with $N = (P, T, F)$.
The *successor decomposition* of a pair of transitions $(x, y) \in >_k^S$ that cannot be concurrently enabled, i. e., $(x, y) \notin ||_c^S$, denoted by $\gg_1^S(x, y) \subseteq >_1^S$, is defined as follows.
For reachable markings $M \in [N, M_0\rangle$ and all firing sequences $\sigma = \langle t_1, ..., t_n\rangle$ enabled in M, i. e., $(N, M)[\sigma\rangle$, with $t_1 = x$, $t_n = y$ and $\forall\ 1 \leq i < j \leq n\ :\ t_i \neq t_j$ it holds that $(t_i, t_{i+1}) \in \gg_1^S(x, y)$ unless $(t_i, t_{i+1}) \in ||_c^S$. ◂

Following the aforementioned predicament of visualizing a successor relation pair of concurrently enabled transitions, we excluded concurrent transition pairs $(x, y) \in ||_c^S$ in the above definition, as there cannot exist a directed path between them in a sound, free-choice workflow system according to Lem. 4.2. To visualize such a relation, we follow the same approach as for bounded net systems: The nearest common ancestor transition $z \in T$ of the concurrent transitions must be found, which is conducted by computing the successor decompositions of the initial transitions of the workflow system with x and y respectively, and, traversing backwards, finding a common ancestor transition. The abstract decomposition of a successor relation only comprises transition pairs in $>_1^S$. Since the definition of $\gg_1^S(x, y)$ only excludes complete loops, it will nevertheless find firing sequences that are part of a loop, if required. Returning to the example, the successor decomposition of $(\mathsf{A}, \mathsf{F}) \in >_{b(S)}^S$ yields $\gg_1^S(\mathsf{A}, \mathsf{E}) = \{(\mathsf{A}, \perp), (\perp, \mathsf{D}), (\mathsf{D}, \mathsf{E})\}$, where $\perp$ refers to the transition without a label.

Computation of successor decomposition

Algorithm 1 shows the computation of the successor decomposition for a transition pair $(x, y) \in >_k^S$ by tracing 1-successor relation pairs, according to Def. 4.13. The algorithm starts with transition x (line 2) and searches for all 1-successors that are in an up-to-$b(S)$-suc-

cessor relation with y (line 8). Transitions, from which 1-successors have been followed, are stored in *visited* to avoid revisiting them (line 8), as this would lead to iterations (see above). The targets of these followed 1-successors are stored in *found* (line 10), from which the search continues in the next iteration (line 17). All found successor transitions are stored in $\gg_1^S(x,y)$. If the 1-successor leads to y it will be added to $\gg_1^S(x,y)$, but the successor transition not further followed (lines 12f). So far, $\gg_1^S(x,y)$ also contains loop-stubs, i. e., 1-successors that do not lead to y without iteration. These are identified and removed in lines 19–23.

Algorithm 1 Computation of the Successor Decomposition.

Input: behavioral relation of a sound free-choice workflow system $>_k^S$ and $||_c^S$; a transition pair (x,y)
Require: $(x,y) \notin ||_c^S \wedge (x,y) \in >_{b(S)}^S$

```
1  visited := {x}
2  next := {x}
3  ≫_1^S(x,y) := ∅
4  while next ≠ ∅ do
5     found := ∅
6     for all n ∈ next do
7        for all (a,b) ∈ >_1^S, such that a = n do
8           if (b,y) ∈ >_{b(S)}^S ∧ (a,b) ∉ ||_c^S ∧ b ∉ visited then
9              visited := visited ∪ a
10             found := found ∪ b
11             ≫_1^S(x,y) := ≫_1^S(x,y) ∪ (a,b)
12          else if b = y ∧ (a,b) ∉ ||_c^S then
13             ≫_1^S(x,y) := ≫_1^S(x,y) ∪ (a,b)
14          end if
15       end for
16    end for
17    next := found
18 end while
19 loop-stubs := ∅
20 repeat
21    ≫_1^S(x,y) := ≫_1^S(x,y) \ loop-stubs
22    loop-stubs := {(a,b) | ∄ c ∈ T : (b,c) ∈ ≫_1^S(x,y)}
23 until loop-stubs = ∅
24 return ≫_1^S(x,y)
```

Algorithm 1 requires, analogous to Def. 4.13, that the transitions of the behavioral relation pair $(x,y) \in >_{b(S)}^S$ cannot be concurrently enabled, i. e., $(x,y) \notin ||_c^S$. To find the common ancestor transition from which successors to x and y shall be projected, Alg. 2 provides an implementation. The algorithm first searches for each transition t that is in weak order with x and y but cannot be concurrently enabled (line 1f) and computes a distance to x and y by the shortest firing sequences to x and y, i. e., the up-to-k-successor relation with smallest

k that contains (t, x) (line 3). From those it chooses the one with the smallest aggregated distance (line 7).

Algorithm 2 Computation of the closest common ancestor of successor relation pair.

Input: behavioral relations of a sound free-choice workflow system $>_k^S$ and $||_c^S$; a transition pair (x, y)
Require: $(x, y) \in ||_c^S$

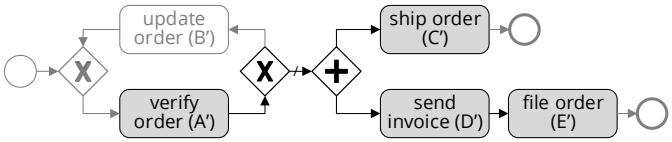
1 **for all** $t \in T$ such that $(t, x), (t, y) \in >_{b(S)}^S$ **do**
2 **if** $(t, x), (t, y) \notin ||_c^S$ **then**
3 $d = \min_{1 \leq k \leq b(S) \ \wedge \ (t,x) \in >_k^S} (k) + \min_{1 \leq k \leq b(S) \ \wedge \ (t,y) \in >_k^S} (k)$
4 $ancestors := ancestors \cup (t, d)$
5 **end if**
6 **end for**
7 **return** t such that $(t, d') \in ancestors \wedge d' = \min_{\{d | (t,d) \in ancestors\}} (d)$

Application to Process Model Search

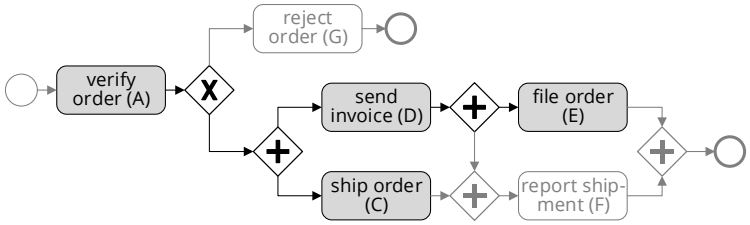
Searching business process models in a large collection stored in process repositories is a prevailing research topic and an abundance of promising search techniques, both structural and behavioral, has been proposed in the literature. Yet, these approaches merely focus on aspects of matching a query with a candidate model and fall short in supporting the reuse of discovered process models.

In this section, we have showed how a successor relation can be projected on the flow relation of a Petri net, by identifying firing sequences that add to the comprehension of behavioral relation pairs and projected them to paths in the net. Interleaved execution semantics may lead to an exponential growth in the number of firing sequences [280] to project. In order to improve the computation of projection for bounded net systems, reduction techniques, such as partial order reduction [279] may be applied. For sound, free-choice workflow systems, we showed that the projection can be computed efficiently, i. e., in $\mathcal{O}(n^4)$ with regard to the number of transitions in the system. Projecting firing sequences founds the basis for various application scenarios in the context of process model management.

This work has been motivated by process model search techniques, i. e., our primary use case is the visualization of matched behavior in a process model graph. To this end, matching has been defined on the set of common successor relation pairs between a query and a candidate, cf. Sect. 4.3. A visualization of the subset of successor relation pairs that are common to both process models, i. e., candidate and query, requires the projection of each successor relation pair, and highlighting the affected paths in the net system. Whereas we defined the projection for net systems, it can be traced back to process models expressed in BPMN, EPC, or UML AD, from which these nets systems have been derived, cf. Sect. 4.2. Figure 26 shows the projec-



(a) Query net system.



(b) Candidate net system.

Figure 26: Intersection of behavioral relations, $>^{S'}_{b(S')} \widehat{\cap} >^{S}_{b(S)}$, projected to BPMN process models.

tion of the intersection of up-to-k-successor relations with successor bound to the business process models from which they have been derived in the first place.

Once the fragment of a process model that matches one or a set of behavioral relation pairs has been discovered, it can also be extracted from the process model and be reused in other scenarios, for instance, as a subsequent step to obtaining matches from a process model collection. To this end, the automatic composition of a process model from fragments of different process models has been proposed in [12, 254]. A query comprises both static and dynamic fragments, whereas dynamic fragments are queries expressed using BPMN-Q [6]. For search, each dynamic fragment is matched against process models from the repository. The fragment of a model that matches the specification of the query is extracted and inserted into the query model in place of the original dynamic fragment. By this, a new model is designed reusing fragments of models stored in the repository in an automated fashion, whereas the new model may comprise a set of fragments that originated from different models. The approaches presented in [12, 254] are limited to structural matching. However, with the projection proposed above, this approach can be extended by behavioral querying.

Finally, the automatic extraction of fragments of process models is relevant for process model refactoring [311]. Here, a means to identify common behavior in various process models offers a major improvement, as it allows discovering process models that contain identical behavior encoded in different graph structures. Our projection enables the extraction of these fragments and streamlining affected process models with a uniform graphical representation of common behavior.

5 QUERYING PROCESS MODELS BY EXAMPLE

This chapter is based on results published in [12, 154, 160].

Querying denotes a search method where the search question comprises only few, yet relevant aspects. In every match, these aspects must be precisely met. In the case of querying process models by example, these aspects are the execution traces characterized by a regular process model. A match must be able to replay each of these traces and may extend them behaviorally. Ranking of matches is guided by the behavioral closeness of a match to the query.

We commence this chapter by a generic definition of process model querying and a brief discussion of use cases that benefit from querying capabilities in Sect. 5.1. This section also introduces an example that is used for clarification of the presented concepts throughout the chapter, followed by a comprehensive discussion of literature on querying process models in Sect. 5.2.

In Sect. 5.3, we present the formal underpinning to deciding a match by a means of behavioral abstraction and show that this can be decided efficiently for sound, free-choice workflow systems using successor relations. Subsequently we introduce a behavioral distance that quantifies the amount of abstraction required for a match, cf. Sect. 5.4. Concluding the chapter, Sect. 5.5 illustrates how these concepts are used to query business process models efficiently.

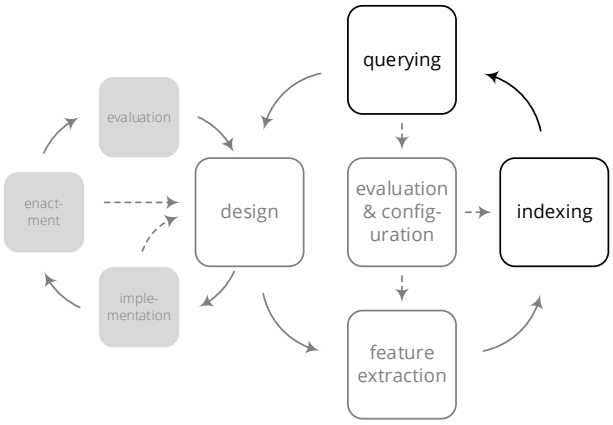


Figure 27: Topics addressed in this chapter: Querying process models, index construction, and efficient search.

5.1 BUSINESS PROCESS MODEL QUERYING

Process model querying enables retrieving process models of which only few, yet important features are known a priori. That is, when users query business process model repositories, they want to find models that considerably add information to the specification of their search question. At the same time, a query expresses essential constraints, i. e., all requirements of the query must be satisfied by a match. This is also referred to as exact or *precise matching*.

Definition 5.1 (Querying Process Models)**.**
Let $\mathcal{C}$ be a set of candidate business process models stored in a repository, and $q \in \mathcal{Q}$ be a query.
Let $\mathcal{F}$ be a set of features, and function $f : \mathcal{Q} \cup \mathcal{C} \mapsto \mathcal{P}(\mathcal{F})$ extract features from a query or a candidate.
Process model querying is a function $\Omega : \mathcal{Q} \mapsto \mathcal{C}^*$, such that

$$c \in \Omega(q) \iff f(q) \subseteq f(c). \qquad \blacktriangleleft$$

In general, a search problem is constrained by (1) the type of data that is searched for, (2) the specification of the search question, and (3) the method to compare individual data instances with the query [333]. Process model search, in general, assumes that (1) is the set of process models stored in the repository. No assumptions need to be made about their representation, as long as their features of interest can be extracted. The query may be expressed in various forms, including textual expressions, e. g., [215, 49], graphical query languages, e. g., [6, 217], or temporal relations, e. g., [268, 228]. Hence, in order to compare candidate and query models that may be of different nature, features of the same type need to be extracted that can then be used to assess the relevance of a candidate for the given query.

Process model querying performs exact matching (3), that is, each feature of the query must also be provided by a matching process model, including negative features, i. e., the absence of features in a candidate requested by the query. At the same time, a match may comprise additional features that have not been requested or excluded by the query. This is closely related with the choice of a perspective for comparison, cf. Sect. 4.3, i. e., the distinctive features of the query are more important than those of the candidates [274]. Following Def. 4.5, the search result $\Omega(q)$ must be ordered by a distance. Due to the aforementioned imbalance of the importance of distinctive features, the distance function employed for ranking may not be symmetric.

Despite the requirement for precise matching, a certain degree of tolerance in matching features can be established by the feature extraction, e. g., similar action labels of query and candidate may be mapped to the same action feature. In Sect. 2.4 it has been required that a search result may not comprise complete models but fragments

or features thereof that have been matched by the query. We consider this as a function of post-processing the search result that can be solved by the projection of successor relations to process models presented in Sect. 4.5.

Use Cases

We briefly recall a variety of scenarios that benefit from querying process models in particular. Process model querying is a central function of business process repositories [331, 49]. For instance, before modeling a new business process, users may search for existing business process models that already solve their problem or contain at least a part of the desired information. Under the assumption that process models stored in a repository underwent some sort of quality control — the design of a business process includes its verification and validation, cf. Sect. 2.1 — querying enables harnessing successful work practice [179]. Once a relevant match has been identified, it can be copied and revised to shorten the design phase [175, 133, 8], which suggests to increase the quality of process models [335]. Eventually, better models that are created more efficiently reduce operational cost [70].

Retrieval & reuse

Alternatively, users may search for auto-completions of a process model [149, 173, 193], or build their process models from readymade templates [51, 101]. In either case, the search question may be a hybrid model which comprises query fragments and static fragments, whereas, upon search, query fragments are replaced with their respective matches [12, 254].

Although the majority of approaches to process model querying focuses on the retrieval of operational business processes [129], these techniques can be used for the discovery and composition of web services, too [265, 214, 261, 184].

Querying can also assist in managing process model repositories, i. e., to detect duplicate fragments [275, 268] or extensions of a process template [180]. Since querying requires a set of features to be met by process models, it has been employed for the analysis of business process models in the context of compliance management [9, 217] and the application of best practices [217, 98], among others. These techniques can even be used for the monitoring of business processes, i. e., queries can discover process instances that do not comply with certain constraints or show particular properties [70, 24, 51].

Process management

Querying Process Models by Example

Process model querying requires a specific language to express certain requirements of the query and map them to features exposed by candidate models. In this chapter, we propose a novel querying approach that has been inspired by Zloof's influential work on *querying*

by example for relational databases [337]. Quite similar, our approach takes a regular process model as input that consists of only few actions and control flow relations between them. This process model acts as an example of the desired behavior, that is, any possible answer must include the example's behavior.

First, we introduce a notion for behavioral inclusion between query and candidate process models, called abstract trace inclusion, and a measure for ranking matching models. Informally, abstract trace inclusion requires that for every trace of the query, a matching model must be able to produce a trace that contains all actions of a query trace in the same order. In other words, a matching model must be able to *replay* any behavior of the query. To rank matching models, we propose an asymmetric closeness measure that quantifies the amount of abstraction required to achieve abstract trace inclusion. Abstract trace inclusion and closeness can be decided for process models that have a finite state space.

Second, we characterize abstract trace inclusion and closeness on the basis of successor relations. Deciding inclusion and computation of closeness can be done efficiently for process models given as sound, free-choice workflow systems. Successor relations also form the basis for an index data structure that enables fast and scalable process model querying.

Example Candidates and Queries

Candidates

We illustrate the topic with an example that founds the basis for later discussions. Fig. 28 shows various business processes modeled in BPMN that handle customer requests regarding the mending of previously sold devices, due to service notifications or customer complaints. All processes include the action manage cost that covers actions with regard to evaluating costs and accounting the correct cost center. Spare parts must be ordered, delivered to the customer, and the warehouse stock must be updated. Also, an invoice is sent to the customer for the provided service. Note that these processes may be initialized by different triggers, i. e., start events, and that the process in Fig. 28c either manages the cost or sends an invoice, depending on the decision, whether the device is still under warranty.

For the sake of clarity, we assume that actions with identical labels refer to the same concept in our examples. We refer to the events and actions by their annotation, i. e., A–F for actions and X and Y for events, hereafter.

Queries

Example queries that outline the expressiveness of our approach are depicted in Fig. 29. Figure 29a presents a purely sequential query, which expresses that all three actions shall be carried out by a matching process in the same order as in the query. This is obviously in line with process models in Fig. 28b and Fig. 28c. The process in Fig. 28a matches this query too, because the parallel gateway allows

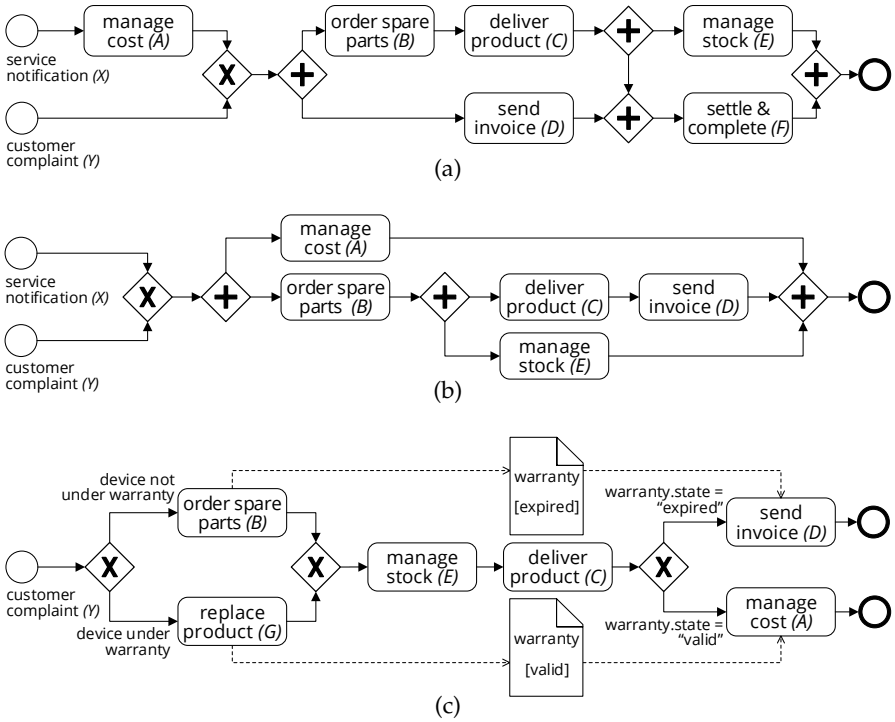


Figure 28: Example candidate process models.

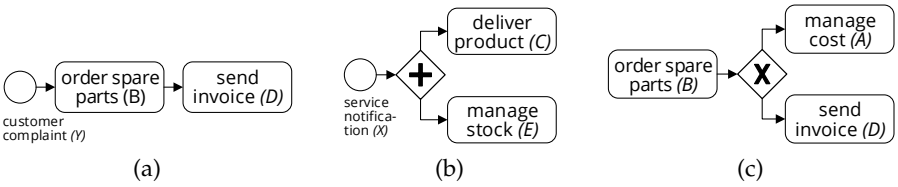


Figure 29: Example query process models.

for the interleaving execution of actions B and D which includes that B can be executed before D. This coincides with the requirement of the query, although both models are structurally distinct from each other, i. e., the sequence flow edge between B and D of the query has no counterpart in the match. The query in Fig. 29b requests a process that allows delivering the product and managing the stock in parallel or interleaving order, i. e., their actual order should not be restricted by a matching process, and Fig. 29c requests models that provide a choice between cost management and sending an invoice after ordering spare parts. We discuss these examples in more detail, in Sect. 5.3.

A comprehensive body of work addresses the computation of similarity between pairs of process models, both by structural and behavioral aspects, cf. Sect. 6.2. However, similarity search of process models is not a promising solution to discover models in our setting, as similarity evaluates how much two process models resemble each other. While either of the example queries has at least one precise

match among the candidates, queries and candidates are not similar at all, in particular, since the queries are much smaller than the candidates. Instead, querying aims at retrieving all models that precisely match a given query, even if the match is a significant extension of the query's features [154].

5.2 RELATED WORK

Querying information in a collection of data items is among the central focus of information systems design. Particularly database systems have embraced the concept to obtain those records that match a certain argument or determine that according records are not to be found [148]. The probably most prominent advancements are the relational database model [54] and the structured query language (SQL), originally proposed in [45]. With the advent of process model repositories, comparable functionality to search process models has received considerable attention in research [331]. Database research not only focuses on returning the correct results, but in particular on data structures and algorithms to organize the stored items in a way that enables fast discovery and retrieval of matches for a query.

Related work on techniques for querying of process models can be classified as follows. The majority of approaches that address structural aspects, i. e., the graph structure of a process model, can be traced back to finding graph homomorphisms [278, 207] between a given query and a candidate model, including graphical query languages that resemble process modeling languages closely and extend them. Semantic approaches capture process models in structured representations and enable reasoning over the various structural aspects. Behavioral query languages generally address temporal properties of the traces of a process model.

Keyword queries

Keyword-based approaches are arguably rather elementary querying techniques: A user enters a set of keywords that are matched with a process' name or description [3, 58]. Often, process models and their elements are annotated with keywords [48] or concepts of an ontology [146, 150] by which they can be queried. In particular in the latter case, ontologies can be used to discover related concepts from a given keyword and obtain according process models. Typically, process models are treated as enclosed objects, and a model must agree with all keywords to be considered a match. In contrast, Liu et al. [175, 259] propose a querying approach that incorporates a hierarchical perspective on processes. Keywords are matched against all actions from a process and its subprocesses and search returns a projection on the minimal subtree of the hierarchy that matches all keywords in a flattened, i. e., non-hierarchic, form.

Structural Querying

The majority of process modeling languages resorts to graph based notations [209] for process models, i. e., nodes and edges of different types. This allows for the application of graph homomorphisms to decide, whether a candidate model matches the structure of a query. A graph homomorphism is a mapping between two graphs such that each node and each edge of one graph, i. e., the query model, has a correspondence in a second graph, i. e., the candidate process model. Consequently, query and candidates are process models that use the same notation.

Graph homomorphism

A fundamental implementation of a graph homomorphism is the subgraph embedding, i. e., if each node and edge of the query can be mapped to a node and an edge in the candidate, than the query is a subgraph of the candidate [41]. Subgraph matching has been proposed for process model querying by Belhajjame and Brambilla [26], who discover corresponding nodes and edges by means of an ontology. Jin et al. [133] address subgraph matching by the definition of a Petri net cover, where the query may consist of several unconnected Petri net fragments. An inverted index based on the paths of all nets allows for fast retrieval of matching results, but limits the alignment to identical transition labels. The notion of Petri net cover coincides with the projection operation proposed by Pankratius and Stucky [229], who introduce a Petri net algebra to manage process models effectively. Although these approaches consider the net's structure, it is clear that identical structures of Petri nets also yield identical behavior with respect to the subgraph. Uba et al. [275] and Ekanayake et al. [82] discover structural overlaps of process models in the form of fragments in a set of process models. They propose an index structure that merges shared fragments and thus allows efficient retrieval of process model clones. It is worth mentioning that these approaches only return the set of matching models but cannot quantify the distance of a match to a query.

Subgraph matching can be relaxed such that edges in the query are either represented as an edge or a path in the graph structure of a matching process model. This increases the flexibility of querying. Lu et al. [180, 178, 179] achieve this by structural reduction techniques, i. e., eliminating all nodes from a candidate that has no correspondence in the query and merging and collapsing edges between the remaining nodes. If the query is a subgraph of the reduced model, it is considered a match. Grigori et al. [105, 104] decide a match by means of an error-correcting subgraph isomorphism [206], i. e., they compute the cost to transform the query such that it becomes a subgraph of the candidate model. This cost is quantified by the graph edit distance [40] and is used to rank the results by their closeness to the match.

Graphical query languages

The reuse of regular process modeling languages and the application of graph homomorphisms are restricted in their expressiveness. In particular, it is impossible to express negated requirements such as the absence of an action or edge in a process model's graph. Therefore, researchers proposed extensions of existing modeling languages or new languages to query for process models in a more expressive fashion. These approaches adopt graph homomorphisms, nonetheless. For instance, PNQL, a query language for Petri nets by Xiao et al. [326], extends the Petri net syntax for queries by transitive, bipartite edges that are mapped to paths in a matching model. A similar extension for BPMN has been proposed by Müller [217]. Additionally, transitive edges may be configured by a minimum and maximum number of nodes that may lie on a path, and nodes of the query can be labeled with regular or ontological expressions to match nodes of a candidate. BP-QL is a graphical query language for BPEL that has been presented by Beeri et al. in [23, 25] and formally analyzed with respect to its expressiveness in [71]. BP-QL acknowledges the hierarchical composition of BPEL processes and provides, besides transitive edges, also compound nodes that are matched against nodes in subprocesses of a candidate.

Awad et al. [6, 8] introduced BPMN-Q, a comprehensive process model query language targeted at BPMN that provides means to express transitive and negative edges, anonymous nodes, generic split and join nodes, and data input and output of actions. A relational decomposition of process model paths enables querying with SQL expressions that are computed from a query, and therefore its fast evaluation. In [10] the authors elaborate on the expansion of queries by substituting labels of the query with similar labels found in the repository to increase the precision of a search result. Yan et al. [330] propose also negative edges, i. e., edges that must not be matched by process models of the search result. This approach is based on dependency graphs that abstract from all nodes but actions and can, therefore, be applied to various process modeling languages. A path-based index allows for efficient exclusion of non-matching candidates during search.

Textual query languages

A number of query languages resort to a textual representation of the request rather than on expressing a query as a graph. The first process model query language was proposed by Christophides et al. [51]. The authors propose a process modeling language that resembles BPEL, which was only proposed later, and use an early XML algebra along with custom functions to traverse the structure of a process model, e. g., to inquire whether a pair of actions is on a sequential path or on parallel paths, or whether they are structurally exclusive. While the language was intended to access properties of a process model that is only represented in XML, it can be repurposed to obtain those models from a repository that satisfy certain structural

aspects. Choi et al. [49] proposed IPM-QL, a query language for their Integrated Process Management platform (IPM), which leverages a custom XML serialization of process models. IPM-QL allows querying for structural aspects that embrace graph homomorphisms, but also for attributes of process model elements. The language has a structure similar to SQL and allows scoping requests to a subset of process models as well as formulating complex queries by means of boolean operators. A similar approach has been followed by Missikoff et al. [214, 265] who propose BPAL as a generic modeling language to capture business processes and QuBPAL as a query language.

The rise of the semantic web has encouraged researchers to apply corresponding techniques to model and manage business processes. Consequently, ontologies have been utilized to capture process models and their relations and generic query languages to reason about these models. In this vein, Ibanez et al. [129] proposed a process model querying approach that uses SPARQL [239] to query process models by structural and quality-of-service aspects. Markovic et al. [193, 191] propose a business process ontology and use WSML [65] to query process models within. While these semantically enriched query languages are expressive, formulation of queries is considered complex and difficult, which limits their applicability considerably.

Behavioral Querying

Querying for structural aspects of the process model graph falls short of recognizing the actual behavior of these models, whereas structurally similar process models may express conflicting behavior, and similar behavior can be expressed with various structural representations. This shortcoming has been addressed by introducing query languages for the behavior of process models that generally focus on temporal relations, i. e., the execution order, of actions. The earliest approach in this respect was made by Momotko and Subieta [215], who proposed BPQL, a textual query language to obtain traces from an event log, i. e., historical instances of a process by temporal relations, such as the execution order of actions, initial and final actions of an instance, but also time constraints, such as the execution duration. Lincoln and Gal [172] use BPMN-Q (see above) to query process models by the life cycles, i. e., state space, of data objects that are manipulated during a process' execution.

Behavioral relations

Temporal logic is used to express properties of a dynamic system with respect to the ordering of events of that system, and is typically applied in the field of model checking [53, 17]. In business process management, temporal logic has been used for compliance management [7], i. e., to prove whether a process complies with certain rules. While approaches to compliance management can be used for querying process models, by using sets of compliance rules as constraints

for a query, we will resort to research work that used temporal logic with the actual intent to query process models.

Shen and Su [261] propose a query language for Petri net systems that evaluates temporal logic formulae, e. g., that an action must be executed before or after another action, or at the beginning or end of a sequence, against the state space of that net system using techniques from model checking. This has been extended by Song et al. [268], who show that LTL [233] constraints can be evaluated against safe net systems by using a finite unfolding [197, 85] of the net that is typically smaller than the net's state space [268]. The authors show that their approach scales well for process model querying in terms of search time.

Behavioral profiles [315] abstract from the behavior of process models and consist of three relations: exclusiveness, strict order, and interleaving order. Any pair of actions of a process model is in exactly one of these relations. These relations have been used by Jin et al. [135] who define BQL, a behavioral query language for process models. A BQL query consists of a boolean composition of pairs of actions in one of the behavioral profile relations. Process models that satisfy all of these relations are considered a match and queries can be evaluated against behavioral profiles of candidate models, which can be conducted efficiently. This approach has been considerably extended by Ouyang et al. [228] who present APQL, a query language that allows for assignments of variables, concatenation of relations, and complex correspondences. Due to its expressiveness, this approach requires evaluation of queries against the state space or unfolding of Petri nets [228].

Behavioral inheritance

Basten and Aalst [19, 281] propose behavioral inheritance of workflow nets as a means to manage process models, e. g., to extract the largest common denominator of a set of process models with respect to their behavior. For that purpose they define four inheritance notions between pairs of process models based on projection and protocol inheritance. If two process models are branching bisimilar [302] under hiding (blocking) transitions of one process model than the second model shows projection (protocol) inheritance with respect to the first model. Although not originally proposed for process model search, either inheritance notion can be used to match a query against candidates, such that a match would be considered a specialization of the query.

Trace matching

Trace semantics declare that the behavior of processes can be captured as the set of traces this process can produce [125]. Consequently, Mahleko et al. [184, 323] argue that a process matches a query, if each trace of the query is a subsequence of one of the traces of the candidate. To avoid infinite sets of traces due to cyclic dependencies, the authors resort to an n-gram representation of traces. If each n-gram of a query can be matched with an n-gram of a candidate, the

candidate is considered a match. Deutch and Milo [69, 70] propose a comparable approach that obtains the k most likely traces of a process model that match a given query trace, where most likely refers to the frequency based on execution probabilities for actions.

Comparison of Approaches & Discussion

In summary, there is a plethora of different approaches for process model querying that ranges from subgraphs to complex query languages on the graph structure and semantic annotations of process models, as well as techniques that aim at the behavior of process models by means of behavioral relations or trace matching. However, the majority of the approaches only focuses on the decision, whether a process model is a match to a given query or not. Requirements such as efficient retrieval or providing distances between a query and matches to assess relevance and derive a ranking are neglected. Also, only few researchers evaluate their approach with regard to quality or performance [155].

Efficient retrieval

For instance, Jin et al. [133] proposed an index for their approach to query by graph homomorphism that is based on computing hashes over all paths of a model. A similar approach is followed by Yan et al. in [330], where paths of models are stored as strings in a database. The approach to trace matching presented in [184, 323] stores n-grams of traces. To quickly prune candidates from search, models that cannot produce all paths or traces of a query are excluded by looking them up in an index. The authors of [275, 82] construct an acyclic graph that stores duplicate fragments as shared nodes.

Although these approaches suggest fast evaluation, they rely on candidates and query using identical labels for actions, which cannot be assumed in general. In [9] a combination of the filtering approach with inexact label matching is proposed that derives additional queries by substituting actions of the original queries with similar actions of models stored in the repository.

Approaches toward behavioral querying suffer particularly from complex evaluation costs, because the resolution of behavioral queries generally requires an analysis of the state space. In a number of cases, finite unfoldings of Petri nets offer an improvement over state spaces with regard to complexity [268], but still suffer from exponential growth [85]. A solution has been proposed by Jin et al. [135], who resort to behavioral profiles to prescribe a set of order relations that shall be matched by a candidate. As behavioral profiles are an abstraction over the traces of a model that neglects causal dependencies, it cannot be guaranteed that several behavioral relations requested by a query can be satisfied by one trace of a model. For each behavioral relation there might be a distinct trace which contains this relation.

Ranking

The majority of techniques only provides a binary answer on a match but do not evaluate how close a matched model is to a given

query. However, to compute a ranking, such a measure that aligns with the concepts to decide a match is required, cf. Sect. 4.3. Approaches toward graph homomorphism [105, 104] propose a distance measure based on the graph edit distance for subgraph homomorphisms [41]. Approaches proposed in [180, 178, 179] compute the closeness of a match and a query by the behavioral precision of shared traces [66] between the query and the reduced models that are discovered as a match.

Query by example

This work has been inspired by *Query by Example* [337], a visual query language for relational databases, where a user provides constant values for some attributes and leaves the others blank. All records that contain these values for the respective attributes are returned in response. Hence, the user proposes an example that becomes completed by the query processor. The simplicity of this approach suggests that also non-expert users become able to query a database [337].

Contribution

We applied this idea to the domain of business processes, and present a novel approach that takes a regular process model as input and retrieves process models from a repository that contain the behavior of the query. As we resort to regular process models for a query, this approach is comparable to graph homomorphisms, but, in contrast, acknowledges the actual behavior rather than static structures. While this restricts queries in their expressiveness, i. e., it is impossible to express the absence of actions or behavior, we argue that it is in line with the targeted use cases, e. g., find process models that contain the modeled behavior, recommend continuations for a model stub, or to detect duplicate fragments. These cases benefit from a simple and intuitive means to express queries more than rich expressiveness that comes at the cost of complexity of queries. This also enables average users to search for process models.

Querying process models by example is an extension of earlier work [154] that decides a behavioral match by the inclusion of precedence relations of a query in the weak order relations of a candidate. We extend this approach by the notion of abstract trace inclusion that establishes a homomorphism between the traces of a query and a matching model. That is, for each trace that can be produced by the query, there exists a trace of a matched model that contains all actions of the query trace and preserves their execution order. Consequently, a match can "replay" the behavior of the query, whereas it may execute additional actions, which yields a more flexible approach than the containment of query sequences in candidate models [184, 323, 70]. It is also different from behavior inheritance [19, 281] that hides or blocks all occurrences of a transition.

A tabular overview of the related work presented in this section is provided in Tab. 13 of Appendix D.

5.3 INCLUSION OF BEHAVIOR

Querying process models by example assumes a regular business process model as a query, i. e., an example of what the process models sought from a repository can do, and, therefore, relies on a notion of behavior inclusion: Whatever behavior the query allows for, shall be supported by a matching model. A match may, nevertheless, extend the behavior of the query. In order to rank the results, i. e., promote more relevant matches to the user, we introduce the notion of closeness that quantifies how much additional behavior a match offers, compared to a query. A model that extends a query less than another model is behaviorally closer and therefore considered more relevant to the user's search intention.

This section presents the formal definition of our approach to behavioral querying. We present a notion of abstract trace inclusion that is used to characterize behavioral matches. We show how this notion is decided for bounded net systems and sound, free-choice workflow systems and elaborate on the expressiveness of the proposed query notion.

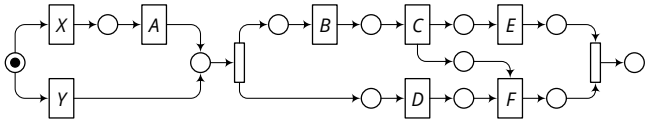
Abstract Trace Inclusion

In this work, we focus on trace semantics of process models, as they are conveyed by their corresponding net systems. Traces have been proposed as a suitable means to capture and compare the behavior of process models [125], which has since seen wide application in the field of process model search [22]. Figure 30 and Fig. 31 show the net systems that correspond to the example process models introduced in Sect. 5.1, following formalization approaches for the control flow [76] and for the data perspective [11]. The latter is needed for representing the data artifacts and their influence on the control flow routing for the model in Fig. 28c.

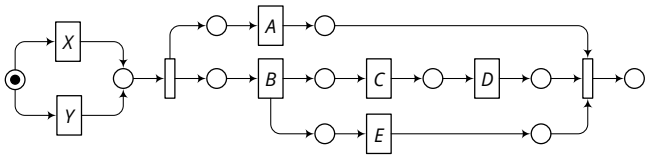
Formalizations of business process models that employ basic control flow routing, such as AND-gateways and XOR-gateways, can be mapped to free-choice workflow systems. As discussed in [177], however, advanced control flow concepts such as OR-gateways and exception handling can typically not be represented with free-choice workflow systems. Also, capturing the data perspective may result in non-free-choice workflow systems. The latter is illustrated by our example: All net systems except for the one formalizing also the data perspective of a business process model, i. e., the net system in Fig. 30c, are indeed free-choice.

Projection of query traces

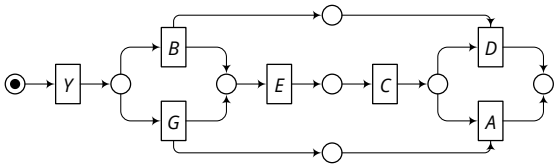
Behavioral inclusion of a query in a matching model implies that each trace a query provides shall be contained in a trace of the matching model, modulo other actions. However, traces of a query may contain transition occurrences that can be misleading. Let $\overline{T}$ be the subset of the transitions of a net system that represents actions in



(a) Net system S_1 of Fig. 28a.

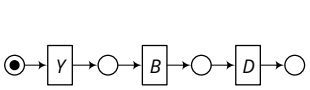


(b) Net system S_2 of Fig. 28b.

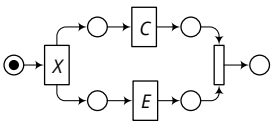


(c) Net system S_3 of Fig. 28c.

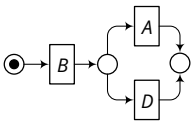
Figure 30: Net systems for example candidates.



(a) Net system Q_1 of Fig. 29a.



(b) Net system Q_2 of Fig. 29b.



(c) Net system Q_3 of Fig. 29c.

Figure 31: Net systems for queries.

the original business process model, i. e., transitions $T \setminus \overline{T}$ have been introduced during the transformation of a business process into a net system to preserve the execution semantics of the process. In our examples, these transitions are depicted without a label. Obviously, these transitions should not be considered for a query. Therefore, we define projected traces that discard all occurrences of transitions that have no business semantics.

Definition 5.2 (Projected Trace)**.**
Let $S = (N, M_0)$, be a net system with $N = (P, T, F)$, and $\overline{T} \subseteq T$.
We define a short-hand notation for transition occurrences of a trace $\sigma \subseteq T^*$ up to index j, such that $\overline{T}^*_{\sigma|j} = \{t_x \in \sigma | x < j \wedge t_x \in \overline{T}\}$. Then, the projection of trace σ by $\overline{T}$ is defined by

$$\overline{\sigma} = \bigcup_{i=0}^{|\overline{T}^*_{\sigma|n}|} (i, t_i)$$

with $t_i \in \overline{T}$, such that $\exists\, j \in \mathbb{N} : (j, t_i) \in \sigma \wedge i = |\overline{T}^*_{\sigma|j}|$. ◂

Consider Q_2 of Fig. 31b, where the final transition has been introduced to obtain a workflow net from the query model. Traces of this

net system contain $\sigma = \langle \mathsf{X}, \mathsf{C}, \mathsf{E}, \bot \rangle$, where $\bot$ refers to the transition without a label. Since this transition conveys no business semantics it shall be disregarded in the traces of Q_2, i. e., $\overline{\sigma} = \langle \mathsf{X}, \mathsf{C}, \mathsf{E} \rangle$. Consequently, the set of projected traces of a net system is denoted by the following notation.

Notation 5.1 (Projected Traces). Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$ and $\overline{T} \subseteq T$. The set of projected traces of a net system is defined by

$$\overline{\mathcal{T}}(N, M_0) = \{\overline{\sigma} \in \overline{T}^* \mid \exists\ \sigma \in T^*, M \in \mathbb{M} : (N, M_0)[\sigma\rangle(N, M)]\}. \quad \blacktriangleleft$$

Abstraction of traces

To actually decide behavior inclusion, we define the notion of loose abstraction for traces. Intuitively speaking, a sequence of transitions is a loose abstraction of a trace, if it is obtained from the trace by removing some occurrences of transitions.

Definition 5.3 (Loose Abstraction)**.**
Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$ and $T' \subseteq T$ a set of transitions. Let $\sigma \in \mathcal{T}(N, M_0)$ be a trace and $\sigma' \in T'^*$ a sequence of transitions.

Sequence σ' is a *loose abstraction* of σ, denoted by $\sigma' = \tau(\sigma)$, iff there exists an injection $f : \sigma' \mapsto \sigma$ such that

- $f((i, x)) = (j, x)$ and $i \leq j$, and
- $f((i, x)) = (j, x)$ and $f((i + 1, y)) = (k, y)$ implies $j < k$. $\blacktriangleleft$

Trace abstraction has been studied from different perspectives in the literature. Loose abstraction is different from the notion of abstraction known in behavior inheritance [19], which abstracts all occurrences of a transition. Stuttering equivalences [17] allow the abstraction of repeated actions if they are not mirrored in a corresponding trace. Loose abstraction, in comparison, allows not only abstraction of repeated, but of any transition occurrence.

Behavioral inclusion is based on loose abstraction in that traces of a net system are an abstraction of traces of another net system. However, in general, these net systems have disjoint sets of transitions as they are defined independently from each other. In order to compare the traces of two net systems, we leverage the correspondence relations $\backsim$ of the alignment of these systems.

Definition 5.4 (Alignment of Firing Sequences)**.**
Let $S = (N, M_0)$ and $S' = (N', M_0')$ be two net systems with $N = (P, T, F)$ and $N' = (P', T', F')$, and $(\backsim, \phi)$ an elementary alignment of their sets of transitions, i. e., $\backsim\ \subseteq T' \times T$. Let $\sigma \in T^*$ and $\sigma' \in T'^*$ be sequences of transitions of equal length, i. e., $l = |\sigma| = |\sigma'|$.

σ' and σ are *equal by alignment* $(\backsim, \phi)$, denoted by $\sigma' \backsimeq \sigma$, iff for all $1 \leq i \leq l$ holds that $t' = \sigma'(i) \wedge t = \sigma(i) \implies (t', t) \in\ \backsim$. $\blacktriangleleft$

Our definition of an alignment of firing sequences is not to be confused with the term "trace alignment" discussed in [35], where a set of historic traces of the same process are aligned with each other by their pairwise similarity. Instead, the term alignment, as it is used here, indicates that each transition occurrence in the trace of one net system has a corresponding transition occurrence in the trace of the other net system by the alignment of their process models, cf. Def. 3.13.

Inclusion of traces

Loose abstraction and the alignment of firing sequences found the basis for abstract trace inclusion, which captures the intuition of behavioral containment of a query in a matching model. It requires each trace of the query net system to correspond to a trace of a matching net system, once a certain set of transition occurrences has been abstracted.

Definition 5.5 (Abstract Trace Inclusion)**.**
Let $S = (N, M_0)$ and $S' = (N', M_0')$ be net systems with $N = (P, T, F)$ and $N' = (P', T', F')$, and $(\phi, \backsim)$ an elementary alignment with $\backsim \subseteq T' \times T$.
S shows *abstract trace inclusion* of S', denoted by $S' \,\widehat{\sqsubseteq}\, S$, iff for all projected traces $\overline{\sigma}' \in \overline{\mathcal{T}}(N', M_0')$ there exists a trace $\sigma \in \mathcal{T}(N, M_0)$ and it holds $\overline{\sigma}' \backsimeq \tau(\sigma)$. ◂

For an example, consider the query model shown in Fig. 29a and its corresponding net system Q_1 in Fig. 31a, which can produce only the trace $\overline{\sigma}' = \langle \mathsf{Y}, \mathsf{B}, \mathsf{D} \rangle$. Net system S_1, depicted in Fig. 30a, can produce the trace $\sigma = \langle Y, \bot, B, \mathsf{C}, \mathsf{E}, D, \mathsf{F} \rangle$, for which $\overline{\sigma}'$ is an abstraction, illustrated by the highlighted transition occurrences. In fact, each net system depicted in Fig. 30 provides at least one trace that can be abstracted to $\overline{\sigma}'$, and therefore, shows abstract trace inclusion with Q_1. Query Q_3, depicted in Fig. 30c, can produce the traces $\langle \mathsf{B}, \mathsf{D} \rangle$ and $\langle \mathsf{B}, \mathsf{A} \rangle$. Whereas the first trace is an abstraction of above trace σ, too, S_1 cannot produce any trace where A is executed after B. Consequently, S_1 does not show abstract trace inclusion of Q_3.

Abstract trace inclusion induces a notion of a match between a query model and some process model. This match is complete in the sense that every trace of the query model is an abstraction of some trace of the matching process model. In contrast, we may speak of a partial match, if abstract trace inclusion holds only for certain traces of the query model. Without further restrictions, however, this notion of partial matching is of limited use. A trace is any firing sequence σ that is enabled in the initial marking M_0 of a net system (N, M_0), cf. Def. 3.4. Even if the empty trace would be excluded, traces may consist of only one occurrence of a transition enabled in M_0. Hence, any candidate model that shows at least a single trace containing one of the transitions enabled in the initial marking of the query would be considered a partial match. Hence, we resort to complete matches.

Deciding Abstract Trace Inclusion

From Def. 5.5 follows that a match is decided by comparing all projected traces of a query with potentially all traces of a candidate. This yields a sufficient condition for abstract trace inclusion. Yet, due to cyclic dependencies, a net system may have an infinite set of traces. As we rely on the net system of a process model, abstract trace inclusion can be decided by means of the state space for bounded net systems, but remains a computationally expensive task. We therefore present a necessary condition that can be evaluated more efficiently based on the successor relations introduced in Sect. 3.3. For sound, free-choice workflow systems, the necessary and sufficient conditions coincide, i. e., abstract trace inclusion can be decided efficiently.

Bounded net systems

Trace equivalence and inclusion are decidable for unlabeled net systems since they are reduced to the reachability problem [112]. Abstraction of traces, however, implies that equivalence and inclusion of traces cannot be reduced to a problem for unlabeled net systems. Hence, we are able to state decidability only for bounded net systems that have a finite state space.

Theorem 5.1.
Abstract trace inclusion is decidable for bounded net systems.

Proof. Since the net systems are bounded, their behavior can be represented by finite labeled transition systems. Each transition in this system may be subject to abstraction. This is incorporated by augmenting the transition system with an additional silent transition per non-silent transition that has the same source and target state. Then, the results follow from decidability of trace inclusion for finite labeled transition systems [280]. □

Deciding abstract trace inclusion for bounded net systems is computationally hard in the general case. In fact, the problem of deciding whether two labeled transition systems show trace inclusion is PSPACE-complete [280].

However, if the successor relations of query and candidate net systems are known, a necessary condition for abstract trace inclusion can be specified based on the containment of correspondences of query transitions in the candidate net system and the notion of successor inclusion. It holds between two net systems, if successorship of transitions in one system implies successorship of the transitions in the other system. The intuition behind is that a query model must not define successorship between two transitions that cannot be mirrored by a matching model even if abstraction is applied. Since there is a successor bound for $b(S)$ successor relations of a net system S, cf. Not. 3.4, successor inclusion requires only the investigation of all relations up to this bound.

	X	Y	A	B	C	D	E	F
X	·	·	1	3	4	3	5	6
Y	·	·	·	2	3	2	4	5
A	·	·	·	2	3	2	4	5
B	·	·	·	·	1	1	2	2
C	·	·	·	·	·	1	1	1
D	·	·	·	1	1	·	1	1
E	·	·	·	·	·	1	·	1
F	·	·	·	·	·	·	1	·

(a) $\trianglerighteq_k^{S_1}$

	X	Y	A	B	C	D	E
X	·	·	2	2	3	4	3
Y	·	·	2	2	3	4	3
A	·	·	·	1	1	1	1
B	·	·	1	·	1	2	1
C	·	·	1	·	·	1	1
D	·	·	1	·	·	·	1
E	·	·	1	·	1	1	·

(b) $\trianglerighteq_k^{S_2}$

	Y	A	B	C	D	E	G
Y	·	4	1	3	4	2	1
A	·	·	·	·	·	·	·
B	·	·	·	2	3	1	·
C	·	1	·	·	1	·	·
D	·	·	·	·	·	·	·
E	·	2	·	1	2	·	·
G	·	3	·	2	·	1	·

(c) $\trianglerighteq_k^{S_3}$

Table 3: Minimal k-successor relations for candidate net systems from Fig. 30.

	Y	B	D
Y	·	1	·
B	·	·	1
D	·	·	·

(a) $>_1^{Q_1}$

	X	C	E
X	·	1	1
C	·	·	1
E	·	1	·

(b) $>_1^{Q_2}$

	A	B	D
A	·	·	·
B	1	·	1
D	·	·	·

(c) $>_1^{Q_3}$

Table 4: 1-successor relations for query net systems from Fig. 31.

Definition 5.6 (Successor Inclusion)**.**
Let $S = (N, M_0)$ and $S' = (N', M_0')$ be net systems with $N = (P, T, F)$ and $N' = (P', T', F')$, $\overline{T'} \subseteq T'$, and $>_{b(S')}^{S'} \subseteq \overline{T'} \times \overline{T'}$ and $>_{b(S)}^{S} \subseteq T \times T$ the successor relations of systems S and S', respectively. Let $(\backsim, \phi)$ be an elementary alignment of the net systems, i. e., $\backsim \subseteq T' \times T$
S shows *successor inclusion* of S', denoted by $S' \,\widehat{\subseteq}\, S$, iff

- $\forall\, t' \in \overline{T'}\ \exists\, t \in T : (t, t') \in \backsim$ and
- $\forall\, (x', y') \in >_{b(S')}^{S'}\ \exists\, (x, y) \in >_{b(S)}^{S} : (x', x), (y', y) \in \backsim$. ◂

As successor inclusion only requires comparing all successors of the query against those of a candidate, it can be computed in quadratic time to the number of transitions in the query net system if the successor relations are known, which is more efficient than computing abstract trace inclusion. Hence, candidate models can be excluded from an exhaustive investigation to decide abstract trace inclusion, if they do not include all successor relationships required by the query.

Theorem 5.2.
Let S and S' be net systems. Then, $S' \not\widehat{\subseteq} S \Rightarrow S' \not\sqsubseteq S$.

Proof. Follows from the definition of abstract trace inclusion and the definition of $(x, y) \in >_{b(S)}^{S}$. From $S' \not\widehat{\subseteq} S$ follows that $\exists\, (x', y') \in >_{b(S')}^{S'}$ $\nexists\, (x, y) \in >_{b(S)}^{S} : (x', x), (y', y) \in \backsim$, and $(x, y) \notin >_{b(S)}^{S}$ ensues that $\nexists\, \sigma \in \mathcal{T}(N, M_0) : (x, y) \notin >_{b(S)}^{\sigma}$, and thus $\nexists\, 1 \leq i \leq |\sigma|, 1 \leq k \leq {}_{b(S)} :$ $\sigma(i) = x \wedge \sigma(i + k) = y$. □

We clarify the necessary condition with the following example. Table 3 and Tab. 4 show successor relations of the example candidate and query models, respectively. In the matrices, we neglect transitions that have no business semantics. For now, we ignore the values of the matrices. Any pair of transitions in a matrix that shows a number is in a successor relationship up to the successor bound of the corresponding net. For instance, Q_3 depicted in Fig. 31c has the successor relation $>^{Q_3}_{b(Q_3)} = \{(\mathsf{B},\mathsf{A}),(\mathsf{B},\mathsf{D})\}$, cf. Tab. 4c. However, none of the up-to-k-successor relations of candidate model S_1 contains the pair (B,A), cf. Tab. 3a. That is, there is no trace such that B is executed before A in S_1. Consequently, S_1 cannot show abstract trace inclusion with Q_3.

Sound, free-choice workflow systems

For the distinguished class of sound, free-choice workflow systems, the notion of successor inclusion, introduced above, is even sufficient to decide abstract trace inclusion.

Theorem 5.3.
Let S and S' be sound, free-choice workflow systems. Then, $S' \overset{\sim}{\sqsubseteq} S \Leftrightarrow S' \overset{\sim}{\subseteq} S$.

Proof.

($\Rightarrow$) Follows from Thm. 5.2.

($\Leftarrow$) By induction on the length of a trace $\overline{\sigma}' \in \overline{\mathcal{T}}(N', M'_0)$.
(Base) Let $\overline{\sigma}' = \langle x' \rangle$. Since S is sound, there is a trace $\sigma \in \mathcal{T}(N, M_0)$ with $(x', x) \in \backsim$ and $\sigma(j) = x$ for $1 \leq j \leq |\sigma|$. Hence, it holds $\overline{\sigma}' = \tau(\sigma)$.
(Step) Let $(x, x'), (y, y') \in \backsim$, $\overline{\sigma}'; y' \in \overline{\mathcal{T}}(N', M'_0)$, $\sigma \in \mathcal{T}(N, M_0)$, and $\overline{\sigma}' = \tau(\sigma)$. Let $\overline{\sigma}'(|\overline{\sigma}'|) = x'$ and, without loss of generality assume that $\sigma(|\sigma|) = x$, as well. Since $S' \overset{\sim}{\subseteq} S$, from $(x', y') \in >^{S'}_1$ it follows that $(x, y) \in >^{S}_{b(S)}$. Let M be the marking with $(N, M_0)[\sigma\rangle(N, M)$. Assume that x is not enabled in M. Then, $(x, y) \in >^{S}_{b(S)}$ implies either (1) $(x, y) \in F^+$ or (2) x and y are enabled concurrently in some marking $M_2 \in [N, M_0\rangle$, cf. Lemma 1 and Theorem 1 in [319].

If (1), then σ can be extended with transitions $t_1, \ldots, t_n \in T$ such that $\sigma; t_1; \ldots; t_n; y \in \mathcal{T}(N, M_0)$, cf. Theorem 1 [319].

Consider (2). Assume $M_2 \notin [N, M\rangle$ for all markings M_2 that enable x and y concurrently. Then, one of those markings M_2 must have been reached after firing $\sigma(j)$, $1 \leq j \leq |\sigma|$, when firing σ in M_0. Since $\sigma(|\sigma|) = x$, it holds $(x, (\sigma(j+1)) \in >^{S}_{b(S)}$, a contradiction with $M_2 \notin [N, M\rangle$. Since $M_2 \in [N, M\rangle$, σ may be extended by firing all transitions $t_1, \ldots, t_n \in T$ to reach M_2, which again yields $\sigma; t_1; \ldots; t_n; y \in \mathcal{T}(N, M_0)$. □

Successor inclusion and, thus, abstract trace inclusion, is decided efficiently.

Lemma 5.1.
Let $S = (N, M_0)$ and $S' = (N', M'_0)$ be sound, free-choice workflow systems with $N = (P, T, F)$ and $N' = (P', T', F')$. Then, $S' \overset{\sim}{\sqsubseteq} S$ is decided in $O(n^3)$ time with $n = \max(|P \cup T|, |P' \cup T'|)$.

Proof. Follows from the computation of k-successor relations of sound, free-choice workflow system in $O(n^3)$ time with n as the number of nodes [314] and Thm. 5.3. □

We conclude this section with a discussion of the expressiveness of queries that results from abstract trace inclusion.

Vocabulary and Expressiveness of a Query

The term vocabulary refers to the concepts of a language by which it is possible to express syntactically correct statements. In our case, the vocabulary used to express a query equals the vocabulary used to express a process model, because we assume a regular process model as query that describes an example of the desired behavior. Evaluation of the query is conducted on an abstraction over the behavioral semantics expressed by this query, that is, execution traces. This abstraction allows searching among process models that have been created with different process modeling languages, and query and candidate model are not required to be expressed in the same modeling language, provided that Petri net formalizations for these languages exist or traces can be computed from the models, e. g., by simulation. For the majority of common business process modeling languages such as BPMN, EPC, UML AD, and BPEL, we rely on net based formalisms, though, cf. Sect. 4.2.

The expressiveness of a query language is restricted by the provided vocabulary and the underlying matching approach. The reuse of common business process modeling languages to express queries and the concept of abstract trace inclusion determine that a query can express the presence of behavior in a match, but not the absence of actions or successor relations between them. We argue that this restriction is in line with the use case we aim for: to enable non-expert users finding process models that provide desired behavior. Query languages that provide a richer expressiveness, such as expressing the absence of concepts, e. g., [49, 25, 268, 228], are more complex and make it difficult to formulate even simple queries.

Abstract trace inclusion for bounded net systems and successor inclusion for sound, free-choice workflow systems, support expressing occurrence of actions, their ordering, and the co-occurrence of several actions in a trace. Each action contained in a query must be contained in a matching model. More precisely, each transition that appears in

a trace of the query must also be contained in at least one trace of the match to hold abstract trace inclusion. The same holds for successor relations. If a pair of transitions appears in any order in a trace of the query, the same order must be mirrored by the successor relation of the match, including repetition of actions. Co-occurrence refers to causal dependencies between actions, i. e., if two actions co-occur in a trace of the query, they must also occur together in a trace of the match. The compliance of these properties in all traces of a match can, however, not be enforced, which is in line with the query being an abstraction of the behavior of a matching model.

Based on this discussion, we investigate matching the example queries with the provided process models. For this purpose, we resort to the basic workflow patterns *sequence*, *parallel split*, and *exclusive choice* [290] that are represented in the example queries in Fig. 29. The corresponding net systems are depicted in Fig. 31.

Sequence

A *sequential path* in a query model, e. g., Q_1 in Fig. 31a, requires a match to provide a trace that executes corresponding transitions in the same order. This may be satisfied, if the corresponding transitions are on a path in the candidate model, as it is the case for S_2 and S_3 in Fig. 30. But, since our approach considers the actual behavior, rather than graph structures, it may also be matched by models that do not show such a path. This becomes apparent in the successor relation of a net, i. e., if a pair of transitions is contained in the successor relation, there exists a trace that contains occurrences of these transitions in the required order. For instance, the concurrent enabling of B and D in S_1 allows for interleaved firing, which includes the order $(\mathsf{B}, \mathsf{D}) \in >^{S_1}_{b(S_1)}$ requested by Q_1. In contrast, S_1 is not a match for Q_3, as it does not allow executing B before A, whereas this is required by the query.

A transition t that is part of a loop of the query is a successor to itself, i. e., $(t, t) \in >^{S}_{b(S)}$, and needs to be matched by candidates, for instance, by cyclic dependencies, too.

Parallel split

Parallel splits allow for the interleaved firing of the parallel actions, as they become enabled concurrently, cf. Fig. 31b. Consequently, the query can produce traces containing any order of execution and successor relations of parallel transitions are symmetric, see, for instance, $(\mathsf{C}, \mathsf{E}), (\mathsf{E}, \mathsf{C}) \in >^{Q_2}_{b(Q_2)}$ in Tab. 4b. Consequently, the ordering of corresponding transitions must not be restricted in a matching model, i. e., the match must also offer symmetric relation pairs. Hence, models S_1 and S_3 of Fig. 30, where actions C and E are strictly ordered, are no matches for Q_2. This is mirrored by the successor relations, cf. Tab. 3, where $(\mathsf{E}, \mathsf{C}) \notin >^{S_1}_{b(S_1)}$ and $(\mathsf{C}, \mathsf{E}) \notin >^{S_3}_{b(S_3)}$. In contrast, $(\mathsf{C}, \mathsf{E}), (\mathsf{E}, \mathsf{C}) \in >^{S_2}_{b(S_2)}$, and hence, S_2 is a match for Q_2 due to the parallel split.

The interleaved ordering of transitions may also be matched by a cyclic structure in a candidate if it contains corresponding transitions, cf. Sect. 4.2.

Exclusive choice

As a query cannot require the absence of a behavioral relation, *exclusive choice* does not express the mutual exclusion of two actions in traces of a matching model. In comparison to sequence and parallel split that require co-occurrence of transitions in at least one trace, this is relaxed for transitions of a query that are exclusive. For instance, transitions A and D of Q_3 in Fig. 31c are not in a successor relation. While it is required that each of these transitions appears in a trace of a match, they are not required to co-occur in a common trace. Nevertheless, co-occurrence is allowed in any order, including interleaving order. For our examples, this ensues that Q_3 matches S_2. Although the exclusive gateway is structurally matched in S_3, this candidate is not a match due to the implicit place between B and D, which renders the net, depicted in Fig. 31b, non-free-choice. Hence, the query trace $\langle \mathsf{B}, \mathsf{A} \rangle$ cannot be mirrored in S_3.

The triggering of a process, in terms of start events, can also be formulated in a query up to certain limitations. Since the transformation of process models to net systems also maps events to transitions, cf. [76, 299, 176], they can be included in the query as well, as it is shown for the queries in Fig. 29a and Fig. 29b. Hence, events are treated similar to actions and can be used for querying as discussed above. Since queries cannot express the absence of behavior, processes that execute actions before these events cannot be excluded, cf. Sect. 4.2.

5.4 CLOSENESS OF INCLUDED BEHAVIOR

Our goal is not only to evaluate queries against candidate process models but also to provide a ranking for matching models, that is, we propose a distance measure for behavioral distance between two models. Since queries typically refer only to a few transitions of the candidate models, this distance needs to be fine-granular and be defined on the level of single transitions. Against this background, our work builds the minimal-k-successor relations as defined in Def. 3.11. This relation captures the minimal distance between the occurrences of two transitions in a dedicated trace, or any trace of a system. As such, they allow for measuring a behavioral distance between models by comparing the minimal occurrence distances for transitions that are part of either model.

Table 3 shows the minimal-k-successor relations for the candidate process models introduced in Sect. 5.1. A value for a pair of transitions in the matrix indicates their minimal distance in any trace of the model. We denote the minimal distance k of a pair of transitions by $(x, y) \in \trianglerighteq_k^S$, hereafter. For instance, A and E of net system S_1 have a minimal distance of 4, i. e., $(\mathsf{A}, \mathsf{E}) \in \trianglerighteq_4^{S_1}$.

Once process models that match a query model in terms of abstract trace inclusion or successor inclusion have been identified, we are

Measuring abstraction

interested in a ranking of these models. This ranking is guided by the amount of abstraction required to achieve abstract trace inclusion, where less abstraction leads to a better match, since the model is closer to the query. The intuition behind the closeness measure can be summarized as follows. Abstract trace inclusion requires that any two transitions that follow each other in some trace of the query model, also follow each other in some trace of the matching model. Still, several transition occurrences may be abstracted in the trace of the matching model. For each pair of transitions in direct successorship in a trace of the query model, we determine the minimal number of transition occurrences that need to be abstracted in some trace of the matching model. This provides us with a measure how *close* the behavior of the model is to the one of the query in terms of single transition pairs.

For an example, consider the trace $\sigma = \langle \mathsf{Y}, \bot, \mathsf{B}, \mathsf{C}, \mathsf{E}, \mathsf{D}, \mathsf{F} \rangle$ of S_1 in Fig. 30a and the query trace $\overline{\sigma}'_{Q_1} = \langle \mathsf{Y}, \mathsf{B}, \mathsf{D} \rangle$ of Q_1 in Fig. 31a. For the successor pair (Y, B) of the query trace $\overline{\sigma}'_{Q_1}$, the occurrence of the unlabeled transition $\bot$ must be removed from σ; for the successor pair (B, D) of $\overline{\sigma}'_{Q_1}$, transition occurrences C and E must be removed from σ. However, S_1 also allows executing D directly after B. In this case, no transition occurrences need to be removed from σ to achieve abstract trace inclusion. Hence, the number of transitions to be abstracted for S_1 equals 1.

Lower closeness values mean that, for a transition pair of the query, more transition occurrences of traces of the match need to be removed to achieve abstract trace inclusion. Closeness is utilized as a ranking score, so that it is applied only when abstract trace inclusion has already been decided. In the same fashion as the decision of abstract trace inclusion, also for the ranking, we first discuss bounded net systems and later turn to sound, free-choice workflow systems which offer the opportunity to compute closeness more efficiently. Finally, we discuss the case of trivial queries comprising a single action.

Bounded net systems

To quantify the amount of abstraction needed to achieve abstract trace inclusion, we first establish the grounding for such a measure by the set of shortest successor-maximal traces. The background for this step is to obtain a set of traces that is finite, but characterizes the essentials of the behavior of a net system. We interpret such behavioral essentials in terms of the 1-successor relation. Hence, we require all traces to be maximal regarding the 1-successor: Any possible extension of a trace must not add information on a new successor dependency. Since there may be an infinite number of such traces, we consider only the shortest trace of those with equal 1-successors.

Definition 5.7 (Shortest Successor-Maximal Traces)**.**
Let $\mathcal{T}(N, M_0)$ be the set of traces of a net system (N, M_0). Then the set of *shortest successor-maximal traces* is defined as $\mathcal{T}_{max}(N, M_0) \subseteq \mathcal{T}(N, M_0)$ such that $\sigma \in \mathcal{T}_{max}(N, M_0)$ iff

- σ is successor-maximal, i. e., for all $x \in T$ holds that either $\sigma; x \notin \mathcal{T}(N, M_0)$ or if $\sigma; x \in \mathcal{T}(N, M_0)$ then $\sigma(j) = \sigma(|\sigma|)$ and $\sigma(j+1) = x$ for some $1 \leq j < |\sigma|$, and
- σ is the shortest of those with equal successors, i. e., $\forall\, \sigma' \in \mathcal{T}(N, M_0)$ such that $\sigma'(j) = x$ and $\sigma'(j+1) = y$ implies $\sigma(l) = x$ and $\sigma(l+1) = y$ for $1 \leq j < |\sigma'|$ and $1 \leq l < |\sigma|$, it holds $|\sigma| \leq |\sigma'|$. ◂

The first condition of the above definition, i. e., σ is successor-maximal, requires that if there exists a trace in the net system that is an extension of σ, this trace must not contain any transition occurrences that add new successor relations to the transitions contained in σ already. From all successor-maximal sequences, the second condition chooses the shortest of those that show identical successor relations. This is, in particular, relevant if the net system contains a loop, and therefore, a potentially infinite set of traces [125].

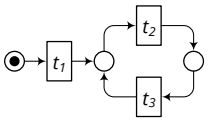


Figure 32: Cyclic net system.

Consider the net system depicted in Fig. 32 that comprises a loop. Due to the cyclic dependency between transitions t_1 and t_3, there exists an infinite set of traces, $\{\langle t_1\rangle, \langle t_1, t_2\rangle, \langle t_1, t_2, t_3\rangle, \langle t_1, t_2, t_3, t_2\rangle, \langle t_1, t_2, t_3, t_2, t_3\rangle, ...\}$. Firing t_2 and t_3 adds new 1-successor relations to traces. However, after the first complete iteration, i. e., when t_2 has been executed the second time, no information is added to the successorship of any of the transitions. Therefore, the set of shortest successor-maximal traces for this net is $\{\langle t_1, t_2, t_3, t_2\rangle\}$.

Using this notion, we define the closeness measure for ranking process models that match a query based on the amount of abstraction required to achieve abstract trace inclusion. Following the above argumentation, we base the measure on pairs of transitions that succeed each other in some query trace and proceed step-wise: First, closeness is defined for a transition pair, grounded on a query trace and a trace of the matching model that induces the smallest amount of abstraction. This is conceptually related to the topic of trace alignment, cf. [35], which seeks to find the optimal alignment between two traces, such that they resemble each other as closely as possible. The closeness between the net systems of a query and a match is then derived by considering all transition pairs that succeed each other directly in some query trace.

Definition 5.8 (Closeness).
Let $S = (N, M_0)$ and $S' = (N', M_0')$ be bounded net systems with $N = (P, T, F)$ and $N' = (P', T', F')$, and $(\backsim, \phi)$ an elementary alignment of their sets of transitions, i. e., $\backsim \subseteq T' \times T$.
Let $>_k^S \subseteq T \times T$, $\trianglerighteq_k^S \subseteq T \times T$, and $>_1^{S'} \subseteq \overline{T'} \times \overline{T'}$ be their up-to-k-successor and minimal-k-successor relations, respectively.
Let $(x', y') \in >_1^{S'}$ be a pair of transitions such that $(x', x), (y', y) \in \backsim$. Their *closeness* in S is defined as

$$c(x', y') = \begin{cases} 0 & \text{if } (x, y) \notin >_{b(S)}^{\sigma} \text{ for} \\ & \quad \sigma \in \mathcal{T}_{max}(N, M_0), \\ & \quad \overline{\sigma'} \in \overline{\mathcal{T}}_{max}(N', M_0'), \\ & \quad \overline{\sigma'} \backsimeq \tau(\sigma) \\ \min\limits_{\substack{\sigma \in \mathcal{T}_{max}(N, M_0),\\ \overline{\sigma'} \in \overline{\mathcal{T}}_{max}(N', M_0'),\\ \overline{\sigma'} \backsimeq \tau(\sigma),\ (x,y) \in \trianglerighteq_k^{\sigma}}} \left(1/\sqrt{k}\right) & \text{otherwise} \end{cases}$$

The overall closeness of S' to S is defined as

$$\rho(S', S) = \sum_{(x', y') \in >_1^{S'}} \frac{c(x', y')}{\left|>_1^{S'}\right|}$$ ◂

In our example, all candidate models depicted in Fig. 28 are matches for the query Q_1 of Fig. 29a, cf. Sect. 5.3. For S_1, $c(\mathsf{Y}, \mathsf{B}) = 1/\sqrt{2}$, due to the trace with the shortest distance between Y and B results in B being a 2-successor of Y. $c(\mathsf{B}, \mathsf{D}) = 1/\sqrt{1}$, because, in S_1, D can be executed directly after B. For the closeness of example models with Q_1 we obtain the following values:

$$\rho(Q_1, S_1) = \tfrac{1/\sqrt{2}+1/\sqrt{1}}{2} \approx 0.85$$
$$\rho(Q_1, S_2) = \tfrac{1/\sqrt{2}+1/\sqrt{2}}{2} \approx 0.71$$
$$\rho(Q_1, S_3) = \tfrac{1/\sqrt{1}+1/\sqrt{3}}{2} \approx 0.79$$

From the net systems in Fig. 30, it becomes apparent that S_1 is in fact behaviorally closest to Q_3 as only transition A needs to be abstracted to achieve abstract trace inclusion. Comparing S_2 and S_3 shows that the closeness measure prefers models with close successors: Although the number of abstracted transitions to achieve abstract trace inclusion is 2 in both net systems, S_3 receives higher closeness due to the direct successorship of (Y, B).

Closeness relates to the amount of abstraction and yields a value in the interval $(0, 1]$. A value of 0 cannot be achieved as all successorships of a query are mirrored in a match, eventually. Closeness should equal 1, if all traces of a query model show trace inclusion with the matching model without any abstraction.

Property 5.1.
Let $S = (N, M_0)$ *and* $S' = (N', M_0')$ *be bounded net systems. Then,* $\overline{\mathcal{T}}(N', M_0') \subseteq \mathcal{T}(N, M_0) \implies \rho(S', S) = 1$.

This property follows directly from the definition of $\rho(S', S)$. For bounded net systems without further restrictions, the reverse does not hold true, though. The measure is based on successorship of transitions, so that, for instance, differences in how often a transition occurs may not be taken into account. Since we apply the measure only for ranking of models for which abstract trace inclusion has already been determined, this aspect is of minor importance. As closeness is a normalized value, a distance function $d_\rho(S', S) = 1 - \rho(S', S)$ can be constructed that provides all required properties for process model search, cf. Sect. 4.3.

Computation of closeness for a given pair of process models imposes challenges. The construction of shortest successor-maximal traces requires investigation of the complete state space of the model, and therefore, has to cope with the state explosion problem [280]. Note, however, that this set may be precomputed for models, because it does not depend on the query model. While this adds to the required space to store process models in a repository, it saves time during search, which we consider more valuable.

Sound, free-choice workflow systems

Again, closeness can be characterized based on successor relations for sound, free-choice workflow systems. We define successor closeness as follows.

Definition 5.9 (Successor Closeness)**.**
Let $S = (N, M_0)$ and $S' = (N', M_0')$ be sound, free-choice workflow systems with $N = (P, T, F)$ and $N' = (P', T', F')$, $\overline{T'} \subseteq T'$, and $(\backsim, \phi)$ an elementary alignment of their sets of transitions, i. e., $\backsim \subseteq T' \times T$. Let $>_k^S \subseteq T \times T$, $\trianglerighteq_k^S \subseteq T \times T$, and $>_1^{S'} \subseteq \overline{T'} \times \overline{T'}$ be the up-to-k-successor and minimal-k-successor relations of systems S and S', respectively.
Let $(x', y') \in >_1^{S'}$ be a pair of transitions such that $(x', x), (y', y) \in \backsim$. Their *successor closeness* in S is defined as

$$c_S(x', y') = \begin{cases} 0 & \text{if } (x, y) \notin >_{b(S)}^S \\ 1/\sqrt{k} \text{ with } (x, y) \in \trianglerighteq_k^S & \text{otherwise} \end{cases}$$

The overall successor closeness of S' to S is defined as

$$\rho_S(S', S) = \sum_{(x', y') \in >_1^{S'}} \frac{c_S(x', y')}{|>_1^{S'}|}$$

◀

Since the successor relation for a net system grows only with the square of the number of transitions in the net, it offers a significant improvement over precomputing and storing shortest successor-maximal traces. Indeed, closeness coincides with successor closeness, for sound, free-choice workflow systems.

Theorem 5.4.
Let S and S' be sound, free-choice workflow systems. Then, $\rho_S(S', S) = \rho(S', S)$.

Proof. Let $S = (N, M_0)$, $N = (P, T, F)$, be a system, $\sigma \in \mathcal{T}(N, M_0)$ with $\sigma(|\sigma|) = x$, and $(x, y) \in \trianglerighteq_k^S$. Let M be the marking with $(N, M_0)[\sigma\rangle(N, M)$ and, without loss of generality, assume that no transition $t \in T$, $t \neq y$, $\bullet t \cap x\bullet = \emptyset$ is enabled in M. If there exist a firing sequence $\sigma_2 = \langle t_1, \ldots, t_{k-1}\rangle \in T^*$ such that $(N, M)[\sigma_2\rangle(N, M_2)$ and $(N, M_0)[y\rangle$ independently of trace σ, then the closeness for a transition pair is independent of the relation between query and model traces. Indeed, the existence of a sequence σ_2 follows from Theorem 35 in [314] stating that $\trianglerighteq_1^S$ and, thus, $>_1^S$ provides a complete characterization of trace semantics of S once the set of transitions $\{t \in T \mid (N, M_0)[t\rangle\}$ enabled in M_0 is known. □

Again, consider the example processes and query Q_1. Net systems S_1 and S_2 are sound, free-choice workflow systems and their closeness to Q_1 is computed purely based on the successor relations. From Tab. 3a and Tab. 4a we observe, for instance that $(\mathsf{Y}, \mathsf{B}) \in >_1^{Q_1}$ and $(\mathsf{Y}, \mathsf{B}) \in \trianglerighteq_2^{S_1}$, and $(\mathsf{B}, \mathsf{D}) \in >_1^{Q_1}$ and $(\mathsf{B}, \mathsf{D}) \in \trianglerighteq_1^{S_1}$, respectively. For the sound, free-choice workflow systems of our example, we obtain the following successor closeness.

$$\rho_S(Q_1, S_1) = \tfrac{1/\sqrt{2}+1/\sqrt{1}}{2} \approx 0.85 \text{ and}$$
$$\rho_S(Q_1, S_2) = \tfrac{1/\sqrt{2}+1/\sqrt{2}}{2} \approx 0.71.$$

The observation that $\rho_S(Q_1, S_1) = \rho(Q_1, S_1)$ and $\rho_S(Q_1, S_2) = \rho(Q_1, S_2)$ is in line with Thm. 5.4. A distance function that leverages successor closeness can be constructed in a straightforward fashion, since the successor closeness is normalized, i. e., $d_{\rho_S}(S', S) = 1 - \rho_S(S', S)$.

In the same manner as for the case of deciding abstract trace inclusion, the utilization of successor relations has the advantage that it allows for efficient computation. Combined with Thm. 5.4, it is possible to decide a match and compute closeness efficiently, if query and candidate are sound, free-choice workflow systems, and only resort to abstract trace inclusion and closeness when they are not. Indeed, for sound, free-choice workflow systems, closeness is computed in low polynomial time to the size of the model.

Lemma 5.2.
Let S and S', $N = (P, T, F)$ and $N' = (P', T', F')$, be sound, free-choice workflow systems. Then, $\rho_S(S', S)$ is computed in $O(n^3)$ time with $n = \max(|P \cup T|, |P' \cup T'|)$.

Proof. Follows from the computation of k-successor relations of sound, free-choice workflow system in $O(n^3)$ time with n as the number of nodes [314] and Thm. 5.4. □

Trivial queries

The aforementioned closeness measures quantify the amount of abstraction needed between pairs of actions or transitions, respectively. Consequently, they are not applicable for trivial queries that comprise a single action only, and for which the successor relation is empty. Matching of trivial queries is captured by the notion of abstract trace inclusion, cf. Def. 5.5. For trivial queries, we argue that an action that is closer to the beginning of a process or its end may be considered more important, as it represents a preliminary step or a result of a process. We incorporate this idea in a closeness measure for trivial queries as follows.

Definition 5.10 (Closeness for Trivial Queries)**.**
Let $S = (N, M_0)$ and $S' = (N', M_0')$ be sound bounded net systems with $N = (P, T, F)$ and $N' = (P', T', F')$ such that $T' = \{t'\}$, and $(\backsim, \phi)$ an elementary alignment of their sets of transitions, i. e., $\backsim \subseteq T' \times T$.

Let $>_k^S \subseteq T \times T$ and $\trianglerighteq_k^S \subseteq T \times T$ the up-to-k-successor and minimal-k-successor relations of S.

Let $T_i = \{t \in T \mid \exists\, \sigma \in \mathcal{T}(N, M_0) : \sigma(1) = t\}$ and $T_o = \{t \in T \mid \exists\, \sigma \in \mathcal{T}(N, M_0), M_0[\sigma\rangle M : \sigma(|\sigma|) = t \wedge \nexists\, t' \in T : M[t'\rangle\}$ be sets of initial and final transitions of S.

For a correspondence $(x', x) \in \backsim$, *closeness for trivial queries* with a transition $y \in T$ is defined as

$$c_0(x', y) = \begin{cases} 0 & \text{if } (x, y) \notin >_{b(S)}^S \\ 1 & \text{if } x = y \\ 1/\sqrt{k+1} \text{ with } (x, y) \in \trianglerighteq_k^S & \text{otherwise} \end{cases}$$

The overall closeness of the trivial query S' to S is defined as

$$\rho_0(S', S) = \max_{t \in T_i \cup T_o,\, t' \in T'} c_0(t', t)$$ ◂

This can be illustrated by a trivial query that consists only of action deliver product (C), which is represented by the net system $Q = ((\emptyset, \{\mathsf{C}\}, \emptyset), \emptyset)$. It is obvious that all example models, cf. Fig. 30, satisfy abstract trace inclusion with Q. To rank these models, we compute the following trivial closeness values. Here, $\perp$ refers to the unnamed final transitions of nets S_1 and S_2.

$$\rho_0(Q, S_1) = \max\left(\tfrac{1}{\sqrt{5}}, \tfrac{1}{\sqrt{4}}, \tfrac{1}{\sqrt{4}}\right) \approx 0.45 \quad (T_i = \{\mathsf{X}, \mathsf{Y}\}, T_o = \{\perp\})$$

$$\rho_0(Q, S_2) = \max\left(\tfrac{1}{\sqrt{4}}, \tfrac{1}{\sqrt{4}}, \tfrac{1}{\sqrt{3}}\right) \approx 0.58 \quad (T_i = \{\mathsf{X}, \mathsf{Y}\}, T_o = \{\perp\})$$

$$\rho_0(Q, S_3) = \max\left(\tfrac{1}{\sqrt{4}}, \tfrac{1}{\sqrt{2}}, \tfrac{1}{\sqrt{2}}\right) \approx 0.71 \quad (T_i = \{\mathsf{Y}\}, T_o = \{\mathsf{D}, \mathsf{A}\})$$

5.5 EFFICIENT QUERYING

Having discussed how a match of a query can be decided and a closeness value be computed for ranking search results, the final section of this chapter elaborates on the application of the presented concepts for efficient querying.

The most simplistic technique to find all process models that match a given query is to compare the query with each model in the repository, decide a match, and compute its closeness to provide a ranking of all models that satisfy the query. The search time of such an approach can be approximated by the product of the number of models in the repository and the average time to decide a match. Hence, it grows linearly to the size of the process model collection and cannot be considered efficient [276].

In order to make search efficient, it is required to avoid exhaustive examination of every process model in the collection [276], and exclude process models that cannot satisfy the matching criterion, early, cf. Sect. 2.4. Since a match is decided on the notion of abstract trace inclusion in the general case, there exists no inherent ordering of process models and traditional indexes, e. g., binary trees, hashing, and spatial access methods in vector spaces, cannot be applied. Also, metric space indexes [276], hierarchical indexes based purely on the pairwise distance between stored data objects[1], cannot be leveraged, as distance functions used for querying to do not satisfy metric properties. Hence, another approach to reduce the computational cost for process model querying may be employed, at the cost of increased storage space.

Index Construction

Problems comparable with the aforementioned have been encountered in the application of graph databases and indexing, e. g., chemical compound data, social networks, or genome data [253]. Such data may not be ordered in a meaningful way and matching a query with candidates cannot be decided purely on metric distance functions, but rather requires complex analyses and comparisons. Research in that field has come up with an elementary solution, known as the *filter and verification* paradigm, cf. [260, 327, 253]. This approach comprises two phases, which we explain in the context of process model search.

Filter & verification

In the first phase, referred to as *filter*, all process models that cannot satisfy the given query are excluded by an inexpensive operation and a candidate set is produced that is significantly smaller than the original model collection. It is important that the filter does not exclude any candidates that could match the given query. In a subsequent

1 Metric trees have been shown to improve similarity search efficiency and are discussed in more detail in Sect. 6.5.

verification phase, each process model of the candidate set is examined exhaustively to decide a match. Additional computation, e. g., to provide a ranking, is only performed in the verification phase. Therefore, optimization of the search performance generally addresses the optimization of the filter phase to be most effective, i. e., exclude as many irrelevant process models as possible, while the used operations should be very fast.

Filter and verification has been frequently applied in the context of process model search, see for instance [180, 15, 10, 133, 8, 134, 136, 329], where the filter typically excludes process models that do not comprise all actions of a query. In the case of querying process models by example, we benefit from the necessary condition of abstract trace inclusion, i. e., all successor relations of a query S' must be included in the successor relations of a matching process model S, i. e., $>^{S'}_{b(S')} \overset{\sim}{\subseteq} >^{S}_{b(S)}$, cf. Thm. 5.2. In order to save computation time, we reduce this to matching only 1-successor relations of a query, i. e., we require that $>^{S'}_{1} \overset{\sim}{\subseteq} >^{S}_{b(S)}$, cf. Def. 5.9. For instance, for a purely sequential query model S', i. e., a process that does not comprise any loops or concurrency, the size of $>^{S'}_{b(S')}$ is $\frac{1}{2} \cdot |\overline{T}| \cdot (|\overline{T}| - 1)$, whereas $|>^{S'}_{1}| = |\overline{T}| - 1$, which improves the efficiency to compare a candidate with the query by up to one order of magnitude. This is also in line with the definition of the closeness measure, cf. Def. 5.8 that compares 1-successor relations of the query with minimal-k-successor relations of a candidate.

Inverted index

Based thereon, we utilize an inverted index to quickly find a candidate set. Inverted indexes store a set of keys and for each key a list of records that feature this key [148]. Here, we choose successor relation pairs as the keys and the associated records are process models that comprise these relation pairs. To construct the inverted index, we examine all process models in a repository. For every process model, we extract all successor relations $(x, y) \in >^{S}_{b(S)}$ and add each pair as a key to the inverted index. If two or more models share a relation, only one key will be contained in the index which then points to a list of these models. By requesting a key, the query allows for quick identification of all process models that contain this particular successor relation. The index also allows for the iterative addition and removal of process models by adding models and keys or removing models from the list of corresponding keys; if a key does not point anymore to at least one process model, it is removed. Table 5 illustrates an excerpt of such an index for the example processes, shown as net systems in Fig. 30, derived from their successor relations, cf. Tab. 3.

Search

Filter

To discover the candidate set in the filter phase, we extract the 1-successor relations from the query by means of its projected traces and use it to find matching relation pairs in the inverted index. That is, we

key	models
(A,B)	$\{S_1, S_2\}$
(A,C)	$\{S_1, S_2\}$
...	...
(B,A)	$\{S_2\}$
(B,C)	$\{S_1, S_2, S_3\}$
(B,D)	$\{S_1, S_2, S_3\}$
...	...
(C,A)	$\{S_2, S_3\}$
(C,D)	$\{S_1, S_2, S_3\}$
...	...

Table 5: Example inverted index.

request the conjunction of these relations which returns a candidate set of models that include all successor relations from the query, and hence, satisfy the necessary condition of a match.

Consider the example query depicted in Fig. 31c. It comprises the following 1-successor relations, which are highlighted in the inverted index, cf. Tab. 5: $>_1^{Q_3} = \{(\mathsf{B},\mathsf{A}), (\mathsf{B},\mathsf{D})\}$. It is easy to see that only model S_2 includes both relation pairs and therefore is chosen as a candidate for the verification phase, whereas the other models are excluded.

For trivial process models, either queries or candidates, comprising only a single action or transition, respectively, we observe that the successor relation is empty. Consequently, the aforementioned index cannot be used for indexing such models and filtering candidates. Note that, according to the notion of abstract trace inclusion, cf. Def. 5.5, answering such a trivial query requires that in a matching model the respective action must be executable. This is a stronger statement than requiring that the action must exist in the process model. To incorporate handling of trivial queries, we additionally store executable actions of all models in the index, which are then used to filter trivial queries.

Verification

For each model in the candidate set provided as a result of the filtering phase, the sufficient condition of a match is verified, i. e., a match is decided with abstract trace inclusion, and its closeness ρ to the query is computed as a score for ranking several matches. Whenever possible, we rely on successor inclusion and the successor closeness ρ_S for sound, free-choice workflow systems, as it coincides with abstract trace inclusion, cf. Thm. 5.4, and allows deciding a match and computing closeness more efficiently than abstract trace inclusion and closeness for bounded net systems, cf. Lem. 5.2. For process models that are not traced back to sound, free-choice workflow systems but bounded net systems, we need to exploit the state space of

the respective models to decide abstract trace inclusion and compute their closeness to the query.

In order to avoid computation of the shortest successor-maximal traces of bounded net systems each time closeness ρ with a query is evaluated, we store the set of these traces for each process model, once it is inserted into the index or updated. As the successor closeness ρ_S can be computed purely by the minimal k-successor relations, cf. Def. 5.9, we also store the minimal k-successor relations of sound, free-choice workflow systems. Finally, for trivial queries comprising a single action, the closeness definition given in Def. 5.10 applies.

Relevance Feedback

The application of an inverted index promises significant improvements concerning the efficiency of search, but imposes a considerable obstacle. Since only successor relation pairs of the query are compared against successor pairs stored in the index, an alignment between the query and any candidate process cannot be constructed in the same fashion that we have introduced in Sect. 3.4. If discovery of relation pairs in the inverted index compares actions by label equivalence, a query may not return any result, although relevant matches are available in the process model repository, due to terminological heterogeneity, e. g., misspellings or the use of synonyms.

In such a case, the querying approach should be able to provide feedback to the user to improve the query and obtain relevant matches. To compensate for terminological heterogeneity, the matching of relation pairs during the filtering phase could employ various string similarity measures, cf. Sect. 3.4, and a threshold that determines a minimum match. However, this leads to a combinatorial problem, as several similar labels may exist among the keys in the inverted index that could be matched to a label from the query.

Query expansion

To solve this obstacle, we briefly elaborate on another solution to the problem that has proven successful in the area of information retrieval [188, 16]. *Query expansion* refers to the concept of creating derived queries by reformulating the initial query to improve the effectiveness of search results [16]. The intuition behind query expansion is that users who have no detailed knowledge of the document collection find it hard to formulate a query that is suited to obtain relevant matches [16]. In the context of process model search, this is particularly problematic with respect to terminology, i. e., the vocabulary used to label process model elements, cf. [92]. Query expansion provides a method for feedback by proposing terms that are related to a given query. Even if a controlled vocabulary has been enforced for process models stored in a repository this allows search to overcome, for instance, spelling mistakes.

Note that we are not interested in reformulating the semantics of the query, i. e., its behavior expressed by the process model, but rather

in reformulating labels of actions using information that is related to the intention of the query. Instead of suggesting related terms for each action label of the initial query individually, we aim at providing a set of derived queries that incorporate these related terms. From our discussion of related work, cf. Sect. 5.2, we conclude that the majority of approaches to process model querying neglects process alignments and therefore cannot compensate for terminological heterogeneity. With regard to query expansion, Jin et al. [135] propose the substitution of query labels by similar labels obtained from the process repository. In [10], related terms are obtained from a process ontology.

Construction of derived queries

Here, we introduce a more comprehensive approach that consists of three steps: (1) identify related terms, (2) choose relevant terms with regard to the query, and (3) create a set of expansions of the initial query. The identification of related terms (1) is based on the similarity of action labels, i. e., each action label of the query is compared with action labels stored in the repository, which exploits the set of executable actions stored in the index to evaluate trivial queries, see above. This may incorporate any of the methods, i. e., syntactic, linguistic, and contextual similarity, introduced in the context of constructing process alignments, cf. Sect. 3.4. All action labels up to a certain threshold may be considered as related terms.

(2) These terms are examined for their relevance to the query, e. g., by studying co-occurrence of related terms in process models stored in the repository. As the inverted index comprises all successor relation pairs of each process model, i. e., pairs of actions that can occur together in a process instance, co-occurrence can be evaluated efficiently. Related terms that do not co-occur with any other term of the initial query in any process model can be safely pruned. (3) Finally, a set of derived queries is computed from the initial query, substituting action labels with identified related terms. This may lead to a large number of derived queries, which can be tackled by computing the similarity of each derived query with the original query and choosing only the top n similar queries, or queries up to a certain similarity threshold. In the next chapter, we elaborate on a similarity measure for process models that also incorporates the similarity of action labels.

The set of derived queries may then be suggested to the user, to choose among them. In [16], it is argued that presenting the user too many suggestions leads to difficulties to choose among them. The above approach reduces the number of relevant terms and derived queries, and mitigates this issue. Alternatively, query expansion can be applied automatically, i. e., without the user choosing from derived queries, which has been proposed in [10]. Then, the presentation and ranking of provided search results must incorporate the similarity of a derived query with the initial query.

6 SEARCHING PROCESS MODELS BY SIMILARITY

This chapter is based on results published in [153, 159, 158, 78].

Similarity search is based on a notion of distance that, given two process models, evaluates how much these models resemble each other, whereas each model may show features that are not met by the other. In this thesis, we compare process model behavior by means of common successor relations. Therefore, we develop a similarity measure that incorporates the quality of a process alignment and allows adjusting the focus on dedicated behavioral aspects, i. e., exclusiveness, strict order, and interleaving order.

In Sect. 6.1, we introduce similarity search for process models in general, illustrate a set of opportunities for its application, and present our running example for this chapter. This is followed by an exhaustive analysis of related work and a comparison of behavioral similarity measures with regard to important properties, cf. Sect. 6.2.

In the following sections, we introduce a set of similarity measures that incorporate the confidence of an alignment, cf. Sect. 6.3, and focus on dedicated behavioral aspects that are combined into a configurable similarity measure for the common behavior of business process models in Sect. 6.4. Finally, in Sect. 6.5, we show how the measure can be applied to indexes that only require a metric distance for efficient search.

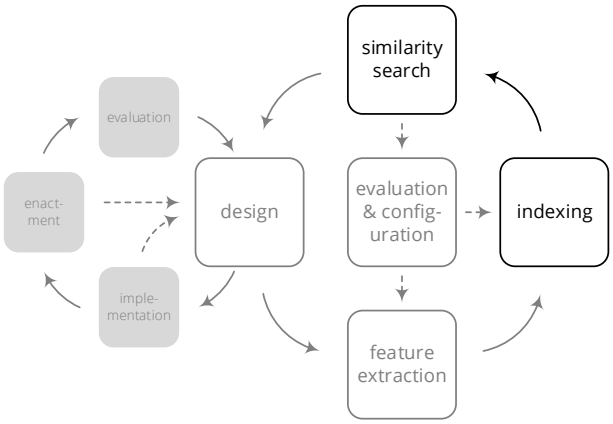


Figure 33: Topics addressed in this chapter: Similarity Search, index construction, and efficient search in metric spaces.

6.1 BUSINESS PROCESS MODEL SIMILARITY SEARCH

Similarity search has been widely addressed in the area of computer science, in particular, in information retrieval [188, 16] where the similarity of documents is of prime interest and has generated an abundance of different techniques [30]. Recently, concepts of similarity search have been applied to various areas that require methods to manage complex and unstructured data, such as multimedia databases, computational biology, or pattern recognition [47], where exact matches are unlikely and for which only pairwise distances can be measured [333].

In the context of business process model management, similarity measures assess how much business processes, specified by respective process models, resemble each other, whereas the majority of research has been devoted to the operational dimensions, that is, actions and their interrelations in a process model, cf. Sect. 2.2. Searching process models by similarity requires a rather complete description of what is searched for, i. e., a regular business process model that can be seen as an example of the information need. A set of process models that share a significant amount of features with the query relative to their distinctive features is returned as search result. The concept of similarity is often expressed by its complement, the *distance*.

Definition 6.1 (Process Model Similarity Search)**.**
Let $\mathcal{C}$ be a set of business process models and $q \in \mathcal{C}$ be a query with a distance threshold $r(q) \in \mathbb{R}$.

Let $d : \mathcal{C} \times \mathcal{C} \mapsto \mathbb{R}$ be a symmetric function that assesses the distance between a query and a candidate.

Process Model Similarity Search is a function $\Omega : \mathcal{C} \mapsto \mathcal{C}^*$, such that

$$c \in \Omega(q) \iff d(q, c) \leq r(q). \qquad \blacktriangleleft$$

We recall that a search problem is constrained by (1) the type of data that is searched for, (2) the specification of the search question, and (3) the method to compare individual data instances with the query [333]. Similarity search in general assumes that (1) candidate and (2) query are of the same type, which is accounted for in above definition by $\mathcal{C}$. For process model similarity, we abstract from the actual modeling language of stored business process models and assume that these models can be captured by a common formalism, e. g., a canonical graph [73, 163], transformations to Petri nets [76, 177], or abstractions of their behavior [300, 158].

Comparing a candidate with the query (3) does not decide on a match directly, as it is the case for querying, cf. Def. 5.1. Instead, only a distance value is computed that represents how much the query and a candidate resemble each other. The search result is then implicitly bounded by a threshold, i. e., the query radius $r(q)$. Moreover,

contrary to querying, not all features required by a query need to be satisfied, if both models are, nevertheless, sufficiently similar. This is the reason, why similarity search is also referred to as *approximate search*. As a similarity query comprises a complete process model for which similar models shall be discovered, there is no clear preference on the perspective for a reference model. Hence, the distinctive features of query and candidate are considered equally important and the distance function needs to be symmetric, cf. [274]. This is in line with literature on similarity search [47, 122, 333].

Use Cases

Similarity search enables a set of different, generic use cases [122]: *Range queries* reveal all process models that are within a certain proximity $r(q)$ of the query; *nearest-neighbor* queries discover the n process models that are most similar to the query. Moreover, similarity search allows *clustering* a collection of process models by their similarity, whereas a cluster contains a set of process models that are more similar to each other than to any other model. A number of concrete use cases for process model similarity and search have been proposed for the management of business process models.

Retrieval & reuse

Retrieval and reuse of process knowledge in the context of designing a new process model is supported by similarity search only to a limited extent, as it presumes an already complete business process model as input. However, in case business processes are modeled in a collaborative fashion, approaches to similarity can assist in identifying commonalities and differences between various versions created by different people [21, 190], or evaluate to which extent models that have been created by different people describe the same business process [334, 128]. With respect to composing web services, which is closely related with business process models [89, 322], nearest neighbor queries can be used to discover alternative services that provide similar or compatible operations [170, 271, 84, 105].

Duplicate detection

Similarity search has its strengths in the management of process model collections, in that it enables the discovery of sets of process models that are similar to each other and may even comprise redundant representations of the same business process [275, 82]. Duplicate process models may originate from different information systems [139], different departments of an organization [300], or as a result of a merger or acquisition of another organization [79]. The discovery and consolidation of redundancy in process models has been reported a tedious and time-consuming task [249] that requires automatic means to identify commonalities among business process models.

Process model similarity therefore provides a means to identify variants of a business process model [178, 334] and create clusters of similar processes that can be managed in a consistent fashion [139,

240, 136]. For instance, a set of process model variants can be incorporated into a common configurable process model [294, 221, 162] or these models may remain separate and a common reference model is derived [88, 97, 139]. In the latter case, variants of reference models can be revealed by searching for models that are similar to the reference up to a certain degree. A means to visualize and extract behavioral commonalities of a pair of business process models has been presented in Sect. 4.5.

Process mining

Moreover, process model similarity has been proposed to assist in process mining, e. g., to find process models in a repository that conforms with a process log [15] or to identify similar process models that have been discovered from separate process logs [2], and thereby assess the similarity of logs.

Searching Process Models by Similarity

Following from the above discussion, there exists a large number of approaches toward process model similarity, which we elaborate on in the following section. Yet, most of these approaches rely on exhaustive searching, i. e., the query model needs to be compared with each model in the repository. Techniques for indexing and efficient search cannot be applied in most cases.

In this chapter, we develop a behavioral measure that quantifies the similarity of process models in terms of successor relations and enables efficient search. Based on the share of commonalities in the successor relations of a pair of process models, we show how the confidence of an alignment is incorporated into the similarity measure to account for inaccuracy or uncertainty with regard to the correspondences of an alignment. In a second step, we construct three elementary similarity measures that evaluate commonalities of more expressive behavioral relations, namely exclusiveness, strict order, and interleaving order, that have been introduced in the context of relation sets [314]. We build on these relations and provide an aggregate similarity that can be configured by individual weights.

Existing work requires the costly manual construction of training sets to draw general conclusions on the configuration of a measure for a model repository. Therefore, we contribute an approach to automatically configure and adjust the metrics toward human assessment that avoids this restriction of applicability. Finally, we show how the developed similarity measure can be used for efficient similarity search with metric tree indexes that require a distance function with metric properties, in particular, the triangle inequality. Although the similarity measure is not a proper metric due to the incorporation of the alignment confidence, it can be used with these indexes due to desirable properties, which we prove for our measure. An exhaustive analysis of search performance and search quality of our behavioral metrics is presented in Sect. 7.3.

Example Process Models

We explain our approach by means of a running example in the remainder of this chapter, which we briefly introduce here. Figure 34 shows two business process models in BPMN that describe processes to solve issues with some devices. A similar case has already been introduced in Sect. 5.1. Upon a service notification or customer complaint, spare parts for the device need to be ordered, delivered and an invoice is sent and settled with the customer. Internally, the risk management addresses means to reduce risks of displeasing customers, and stock management concerns the replenishment of supplies.

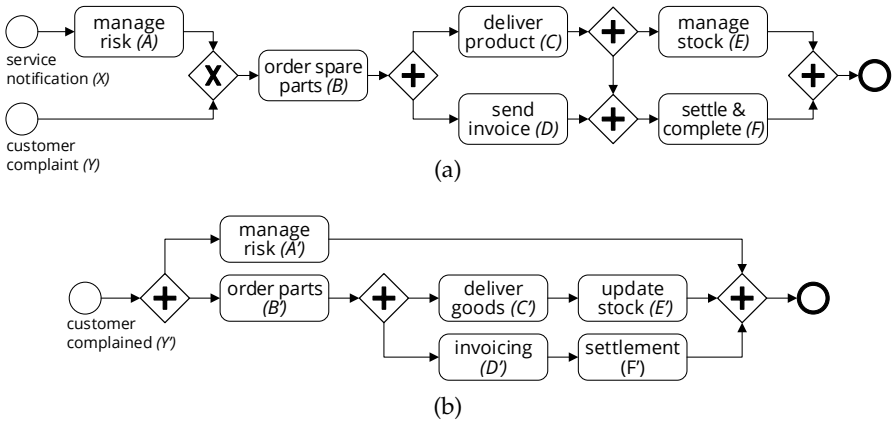


Figure 34: Two similar BPMN process models.

Note that labels of the actions are not equal in both processes, but resemble each other to some extent. We introduced this inaccuracy intentionally to show that an alignment can be constructed on similar actions and that this inaccuracy should be incorporated into the overall similarity of process models. While we elaborate on the construction of an alignment for these processes later, we already point out correspondences by matching character annotations. For instance, order spare parts of Fig. 34a and order parts of Fig. 34b likely represent the same task in practice, indicated by the character annotations B and B′. To keep discussions concise, we refer to actions and their labels by these character annotations, hereafter.

Neither of these models can be used for querying one another, cf. Chap. 5, as both processes show features that cannot be met by the other process model. For instance, risk management is carried out in all instances of Fig. 34b, whereas it is only performed if Fig. 34a is started with a service notification. While settlement can be performed directly after an invoice has been sent to the customer in the former process, it depends on the delivery of the product in the latter.

Before we proceed to the presentation of our approach to process model similarity search based on common successor relation pairs, we examine approaches to process model similarity that have been presented in literature. We compare them with respect to the prop-

erties for distance measures introduced in Sect. 4.3 and discuss the proposed measure in the context of related work.

6.2 RELATED WORK

The topic of process model similarity has been studied from a number of perspectives. Approaches that examine process models for equivalence, e. g., trace equivalence [124, 125], workflow inheritance [287], and bisimulation [211, 301], only tell, whether two workflows are equal. Thus, more expressive notions for process models have been demanded, e. g., in [66]. Dumas et al. [79] provide an overview of general principles to process model similarity search and conclude that similarity typically addresses the structure of the model graph or the process' behavior. This is in line with [316].

A comparative survey of a set of process model similarity measures is presented in [22], which evaluates the agreement of these approaches toward the similarity of process models experimentally. This section explores structural and behavioral approaches that address process model similarity, of which a part has been covered in [22] already. In contrast to [22], we will only briefly discuss structural similarity and put the focus on behavioral similarity as this is the topic of our research.

Structural Similarity

Structural approaches to process model similarity are restricted to properties of the model graph, i. e., nodes, edges, and various attributes, such as involved roles, input and output data, or involved information systems [72, 322]. Therefore, related work to structural process model similarity generally relies on solutions presented for graph-based pattern matching [41, 46] including aggregated similarities or vector similarities over a set of features [256, 86, 16], common subgraphs [199, 42], or graph edit distances [40, 96].

An early solution to process model similarity that uses similarity flooding [201] has been presented by Madhusudan at al. [183]. Based on an initial alignment of actions, the similarity of each corresponding pair of actions, i. e., their confidence, is refined incorporating the similarity of its direct predecessor and successor actions, which in turn updates the similarity of the correspondence. This is repeated, until the similarities of the correspondences converge within a sufficiently small interval, rspectively. Process similarity can then be derived from the final similarity of the correspondence pairs as this incorporates the process model structure.

Feature aggregation

Features are characteristic attributes extracted from a process model such as its modeling language, but also the set of actions it contains, and similarity can be typically estimated by the share of common features. This requires domain-specific similarity functions for different

feature types. Humm and Fengel [128] and Akkiraju and Ivan [3] aggregate distinct similarity functions for various feature types, such as process modeling language, name of the processes, and common actions or model fragments. The similarity function proposed by Ehrig et al. [81] measures the share of common actions, refined by the semantic similarity of these actions that is derived from the confidence of the alignment, cf. Sect. 3.4. Complementary, common flow relations between actions are used for similarity assessment by De Medeiros et al. [66, 293]. Bergmann and Gil [28] consider both, common actions and edges of the process model graph, whereas they account for the latter by predecessors and successors of aligned actions.

Rather comprehensively, Gao et al. [94, 95], define similarity functions for labels of actions, their predecessors and successors, data in- and output, and involved resources. Similarity of two process models is then computed by a similarity function that iteratively converges to an optimal alignment of the actions by the similarity of these features. The approach followed by Melcher and Seese [200], in contrast, does not compare process models at all, but computes process model quality measures over various structural aspects, such as its size or the number of splits or joins. These measures have been introduced in [202]. The authors argue that two models are similar if they have similar values in these measures, but this approach completely disregards the actual actions that are defined in the model.

Feature vectors

An orthogonal approach to compare features without the definition of similarity functions for particular feature types is the computation of a distance in high-dimensional vector spaces [256], where process models are represented as *feature vectors*. For instance, Hornung et al. [126] annotate business processes with a number of tags. A search query consists of a set of tags as well, and similarity is evaluated by the cosine similarity [255] of the feature vectors of a candidate and the query, respectively. However, the annotation of process models with tags needs is carried out manually and requires time and effort. To overcome this issue, Jung et al. [137] define a vector space with dimensions for each distinct action and each path between actions, of any process models. If a process model contains an action or a path, its feature value for that dimension is 1, otherwise 0. This is extended with execution probabilities of actions, and traversal probabilities of paths in [139, 138]. Similarity of two process models is computed by the Cosine similarity function that measures the angle between the vectors [255]. Esgin and Senkul [83] construct feature vectors of actions and their predecessors and successors, and compute similarity by the Euclidean distance.

Another vector representation is chosen by Bae et al. [13], who decompose a process model into structured blocks, i. e., sequences, loops, and concurrent and alternative paths, which can be nested.

Nested structures are represented as a binary tree, and the vector captures all triples of a branch as one dimension. The similarity of two models is then computed by the Manhattan distance [151] of the two corresponding vectors. In further work, Bae et al. [15, 14] resort to incidence matrices which capture paths between actions in a square matrix over the set of these actions. The distance of two process models is computed by subtraction of the incidence matrices of two process models.

Common subgraphs

All of the above approaches to feature similarity are restricted to sets of actions and dependencies between them, i. e., they disregard control flow nodes such events and gateways, and do not account for common fragments in the graph structure of two process models. However, as process models are typically represented by graphs [209], it has been argued that two process models should be considered similar if they share non-trivial fragments of the model graph, i. e., *common subgraphs*.

Cuzzocrea et al. [61] compute the largest common subgraph [42] of a given query and candidate model, and compare it to the size of the query. Corresponding nodes of the subgraph must be of the same type, and the alignment is constructed by means of an ontology over the labels of actions. If the common subgraph is equal to the query, the model is considered a perfect match. While Niemann et al. [223] restrict subgraphs to fragments with only one entry and one exit nodes, they discover all common subgraphs of a pair of models, which they refer to as related cluster pairs. The larger related cluster pairs are, the higher is the similarity of compared process models. A similar approach followed by Ma et al. [181], incorporates also hierarchical process models, i. e., subprocesses, in the comparison of subgraphs. However, this approach is limited to BPEL. Jin et al. [134] transform a process model into a task edge graph that represents paths in the process model. The largest common subgraph of the task edge graph captures common paths in a pair of process models. The size of the maximum common task edge subgraph compared to the size of the smaller task edge graph of either process model yields a structural similarity.

Whereas above approaches compute common subgraphs, Yan et al. [328] extract characteristic fragments, i. e., sequences, loops, splits, etc., and correlate them with the help of the process alignment. As this yields only a coarse estimation of similarity, it is employed in later work of the authors as a preprocessing step for process model search to filter candidates, whereas the actual similarity is computed by the graph edit distance [329].

Graph edit distances

Graph edit distances have received broad acknowledgement in pattern matching and recognition [96] and have been shown to be an efficient means to establish graph isomorphisms [42]. The graph edit distance originally presented by Bunke and Allermann [40] gener-

alizes the string edit distance [168] and measures the difference between two graphs by the minimum cost to transform one graph into the other by operations for the deletion, insertion, or substitution of nodes and edges.

The graph edit distance has been confirmed for its suitability for process model similarity by Dijkman et al. [75, 73, 74]. Due to its exponential computation complexity [40], the authors compare various heuristics and optimizations, and conclude with good effectiveness and efficiency of their approach for process model similarity. Results of this work have been used for process model clustering in [240] and in [82] to rank matches of a query. Its applicability for efficient search has been demonstrated in [153]. For the computation of the graph edit distance, the process model is reduced to a dependency graph that only consists of actions and edges between them, as other nodes may be unlabeled, which makes computation of the graph edit distance hard. The same issue has been faced by Minor et al. [213], who, in contrast, propose various approximations and heuristics to incorporate also control flow nodes that have no explicit label into the graph edit distance.

While graph edit distances determine the cost for edit operations, Küster et al. [161] propose an approach to compute the actually required edit operations and provide a change log between two versions of a process model. Li et al. [169] argue that atomic edit operations impose an unbalanced degradation of similarity for process models and therefore define high-level change operations, e. g., moving the position of an action within the process, and compute graph edit distance by custom weights for these operations.

Behavioral Similarity

Techniques that evaluate behavioral similarity do not only incorporate the structure of the process model graph, but rather the potential or observable behavior that the model, i. e., the graph plus the semantics of its elements, prescribes. In the following paragraphs, we show that related research to behavioral process model similarity relies on trace semantics [125], either by comparing execution traces of a pair of process model directly, or leveraging abstractions thereof that have been presented in Sect. 4.1.

Similarity Based on Execution Traces

A number of authors have proposed business process and web service matchmaking techniques that provide only a binary answer, i. e., two services are considered a match or similar, if they share at least one common transition sequence [325, 184, 170, 190]. However, these techniques do not provide any measure of how similar two process models are, and will not be considered further, hereafter.

Common traces

A straightforward approach to quantify the similarity of process behavior is the computation of the number of *traces that two processes have in common*, presented by De Medeiros et al. [66, 293], who relate this number to the number of traces of either model, and Zha et al. [334], who relate this to the cardinality of the union of trace sets of both models. However, if either of the compared process models has an action contained in each trace that is not contained in any trace of the other model, their similarity is 0. This, along with the problem of infinite sets of traces, has been acknowledged by the authors, who leverage these measures only as a reference without practical application. The problem of actions that are unique to either process model has been addressed by Li et al. [170] by means of trace projections. Prior to deciding trace equivalence, the authors remove all occurrences of actions that do not occur in the traces of both process models. However, their approach provides only a binary answer on similarity.

De Medeiros et al. [66, 293] extend this notion with actual log entries, i. e., only consider the similarity of traces that represent historically executed process instance. By adding frequency distributions of traces, the authors provide a similarity measure that incorporates the relative importance of sequences. This has been leveraged in [180, 178, 179] to rank previously discovered process model variants with regard to a given query. Becker et al. [21] compare the relative frequency distributions of equivalent traces of two processes to compute their similarity. If a trace cannot be produced by a process, then it has a frequency of 0, and two traces show maximum similarity, if they appear equally often. However, it needs to be understood that these approaches require a process log, which cannot be assumed in the general case.

Common firing sequences

Another solution to the problem of actions owned by one process only is the discovery of *common firing sequences*, i. e., fragments of the traces of a process model. Gerke et al. [97] find, for each trace of one process model, the longest common firing sequence in a trace of the other model and relate this to the length of the trace, which yields a similarity measure. Then the compliance of one process model with regard to another is computed by the similarity of each trace of that model with the traces of the second model, which results in a non-symmetric similarity function. Infinite traces are prevented by cutting off iterations. Contrarily, Wang et al. [307] compute principal transition sequences that isolate iterative from non-iterative sequences in a trace. Similarity of two firing sequences is computed by the size of longest common subsequence relative to the size of the longer of both sequences. Similarity of a pair of process models is computed by the average similarity over all principle transition sequences of these models.

Edit distance for traces

Traces are essentially sequences of labels, i. e., symbols that represent actions, and therefore, can be treated as strings. Consequently, the distance between a pair of traces can be computed by the *string edit distance* [168]. For instance, Sun [271] computes the string edit distance over pairs of traces and incorporates frequency distributions from process logs to account for the relative importance of certain traces and to cut off iterations. Again, the availability of logs cannot be assumed in the general case. Alternatively, Wombacher et al. [324, 323] split a trace into sequences of distinct n-grams that are ordered by their occurrence in the trace. This also eliminates iterations and reduces the required space to capture interleaved execution, as recurring n-grams are discarded. The difference between two traces is then computed by the edit distance of the n-gram sequences. To judge on the similarity of a pair of process models, n-gram sequences of both models are aligned pairwise in a way that reduces the cost determined by the edit distance, and process model similarity is computed by the average of distances over corresponding n-gram sequences.

Similarity Based on Behavioral Relations

Due to the state space explosion problem [280], similarity measures based on traces suffer from exponential computation time, independent of solutions that avoid infinite sets of traces due to iterations. Therefore, solutions that mitigate that problem, i. e., abstractions of traces by behavioral relations, have been proposed for process model similarity. This is significant for process model search, because search within large collections is, in particular, sensitive to computational complexity [333]. We have introduced the concepts behind behavioral relations in Sect. 4.1, already.

Causal footprints

Causal footprints store, for each action, sets of actions which are causally related as indirect predecessors and successors, from which footprint vectors can be computed. The similarity of these vector representations of causal links is computed by means of the cosine similarity [300, 205]. This approach has been evaluated for its good approximation of a user assessment of process similarity, but showed high computation times [75]. This is due to the definition of footprints that relates each action with a set of subsets of the process' actions, i. e., footprints have a size exponential to the size of the number of actions in a process. It is noteworthy that [300] was one of the first and most influential works on business process similarity and process model search [78].

Precedence relation

Similarity based on *precedence relations*, i. e., pairs of directly succeeding actions in a process' trace, has been proposed for process model search by Zha et al. [334], who compute similarity by the Jaccard coefficient [132]. Aiolli et al. [2, 1] extend this approach by negative relations for pairs of transitions, i. e., contradictions that a

model shall not violate. However, these relations hold only for the restricted class of well-structured and sound process models and examples show that the distance leads to invalid results in other cases [1]. In compliance with Def. 3.9, we refer to the precedence relation also as 1-successor relation, hereafter.

Weak order relation

Weak order relations are an extension of precedence relations, in that they are not restricted to pairs of transitions that can only be executed directly after one another, but also indirectly. As a consequence, weak order relations can be used to reason on the behavioral relationship of actions in the context of the whole process. This has been used by Eshuis and Grefen [84] to derive profile relations that express whether two actions that occur in at least one process instance, are always executed in the same order, in interleaving order, or are mutually exclusive [319]. Whereas the approach in [84] is restricted to measuring similarity of well-structured processes expressed in BPEL, we have proposed a generalized version [158, 159] that covers processes with a finite state space. In [158], we defined a configurable process similarity metric that has been evaluated for its good approximation of human assessment toward process model similarity. Thanks to its metric properties, it can also be leveraged for metric tree indexes in order to increase the performance of process model search [153]. The weak order relation of a system S coincides with the up-to-k-successor relation, $>_k^S$, with successor bound, i. e., $k \geq b(S)$, cf. Def. 3.10.

Comparison of Approaches & Discussion

Properties of distance functions

We conclude properties for behavioral distances from related work in a similar vein as they have been defined in Sect. 4.3 and list them in Tab. 6 for comparison. In the table, *scale* identifies the codomain of the similarity or distance function, where $[0, 1]$ identifies a normalized real value between 0 and 1, and $\mathbb{R}$ a distance value that is not normalized. *Reflexivity* requires that the distance of a process model to itself is 0, and *monotonicity* expresses that a model pair is more distant the less commonalities or more differences it exposes. All approaches are *non-negative*, reflexive, and monotonic.

Some approaches that address similarity do not show *symmetry* because they compare the commonalities only with the capabilities of one of the compared models, similar to precision and recall presented in Sect. 4.3. Although we required symmetry for similarity, these approaches provide an inexact matching of a query with candidate models and the distance or similarity functions can be turned into a symmetric pendant, for instance, by computing the distance in both directions and computing the arithmetic mean of both values.

Identity refers to the property that two systems are equivalent, if they have a distance of 0. As all the above approaches to behavioral similarity are based on trace semantics, identity applies to trace equivalence. It is obvious that non-symmetric approaches cannot satisfy

APPROACH	TYPE	SCALE	NON-NEGATIVITY	REFLEXIVITY	SYMMETRY	MONOTONICITY	IDENTITY	TRIANGLE INEQUALITY	ALIGNMENT
De Medeiros et al. [66, 293]	traces (logs)	$[0,1]$	✓	✓	–	✓	–	–	–
Li et al. [170]	traces (PN)	$\{0,1\}$	✓	✓	✓	✓	–	–	–
Becker et al. [21]	traces (logs)	$\mathbb{R}$	✓	✓	✓	✓	✓	–	–
Gerke et al. [97]	traces (PN)	$[0,1]$	✓	✓	–	✓	–	–	✓
Wang et al. [307]	traces (PN)	$[0,1]$	✓	✓	✓	✓	✓	–	–
Sun [271]	traces (PN)	$[0,1]$	✓	✓	✓	✓	✓	–	✓
Wombacher et al. [324, 323]	traces (logs)	$\mathbb{R}$	✓	✓	–	✓	–	–	–
Dongen et al. [75, 300, 205]	relations	$[0,1]$	✓	✓	✓	✓	–	✓	✓
Zha et al. [334]	relations	$[0,1]$	✓	✓	✓	✓	–	✓	–
Aiolli et al. [2, 1]	relations	$[0,1]$	✓	✓	✓	✓	–	✓	–
Eshuis and Grefen [84]	relations	$[0,1]$	✓	✓	✓	✓	–	✓	–
Kunze et al. [158, 159]	relations	$[0,1]$	✓	✓	✓	✓	–	✓	✓

Table 6: Overview of related work toward similarity with regard to process model behavior.

identity. On the other hand, none of the approaches that rely on behavioral relations satisfies identity in the general case, as behavioral relations are an abstraction over the traces of a net system. However, it has been shown in [314] that for sound, free-choice workflow systems, trace equivalence holds, if two process models have equal up-to-k-successor relations for all k up to the successor bound $b(S)$, cf. Not. 3.4.

The *triangle inequality* addresses a property of distance functions that provides a notion of transitivity and will be discussed in more detail in Sect. 6.5. Approaches that map models into a vector space and measure distances therein, e. g., [300, 205], obviously provide this property. The use of the Jaccard coefficient to measure the similarity

of two overlapping sets also yields a metric [174], and therefore satisfies the triangle inequality. Related approaches on the similarity of behavioral relations leverage this coefficient, and hence, provide measures that comply with the triangle inequality. However, this is based on the assumption that the sets of actions of two process models are overlapping. All approaches that rely on behavioral relations to evaluate process similarity resort to an abstraction, where correspondences of a pair of process models are treated as identical actions, which yields an overlap of these sets. However, these approaches cannot incorporate the confidence of an alignment. Only Sun [271] incorporates the confidence of a process alignment into the evaluation of process model similarity by comparing traces. Dongen et al. [300] define a separate similarity based purely on the alignment of process models and combine this with the vector similarity for a pair of process models by means of a weighted sum.

Few approaches addressed the problem of searching efficiently, i. e., leveraging an index to avoid examining all models stored in the process model repository [276]. Wombacher et al. [324, 323] propose using standard techniques, e. g., a relation database that provides full-text indexes and store n-grams of the firing sequences as index terms. In contrast, the approach presented by Yan et al. [329] applies the filter and verification paradigm, presented in Sect. 5.5, and proposes a computationally inexpensive function to exclude candidates that are sufficiently dissimilar to the query. Instead of the construction of a separate index, Uba et al. [275] manage process models, such that common fragments are only stored once and are associated with processes that share these fragments as duplicates.

Contribution

The approach to process model similarity based on a behavioral relation distance that is presented in this thesis, is based on [158], whereas it provides a generalization of techniques that incorporate the precedence relation [334, 1, 2] and the weak order relation [84, 158], as these relations are only the ends of the spectrum of up-to-k-successor relations. Similar to [1, 2, 158], we propose a configurable measure that offers weights to prioritize behavioral relations: strict order, exclusiveness, and interleaving order. To avoid guessing these weights or determining them with a training set, we propose a mechanism to learn such weights from a given training set of process models.

As an extension to earlier work [158], we incorporate the confidence of correspondences into the similarity measure and show that this can be well integrated with metric space search principles that have been applied earlier for structural process model search [153].

A tabular overview of the related work presented in this section is provided in Tab. 14 of Appendix D.

6.3 PROBABILISTIC BEHAVIORAL SIMILARITY

Successor relations, as introduced in Sect. 4.2, are an abstraction of behavior in terms of the traces a process model can produce. The abstraction focuses on the potential execution order of actions and trades other behavioral details, such as causality and cardinality for a finite and compact representation of the behavior. While the precise behavior of a process model captured by a state space grows exponentially with the number of actions due to concurrency [280], the size of a successor relation is always upper-bounded by the square of the number of observable actions. Findings from experiments with process variants [319] and empirical work on the perception of consistency between process models [313] suggest that relation sets, an extension of successor relations, capture a significant share of the behavioral characteristics of a process model. This is in line with [158], where a prior evaluation of process model similarity based on relation sets provided a good approximation of human assessment. This provides us with the confirmation that the information loss implied by the abstraction of successor relations is suited to be traded for computational efficiency. Moreover, the compact representation of behavior in terms of finite relations allows for the definition of a similarity measure that enables efficient search.

Suitability of successor relations for similarity

A process model resembles another model in certain behavioral aspects if they overlap in their successor relations. Consequently, the intuition behind a successor distance is to compare the successor relations by corresponding actions and to quantify commonalities and differences, i. e., distinct relation pairs of either model. Therefore, we adopt the Jaccard coefficient [132], $sim_{\mathcal{J}}(A, B)$ as an established measure for the similarity of sets A and B.

Jaccard coefficient

$$sim_{\mathcal{J}}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Subtracting this coefficient from one directly yields a distance metric [174], $d_{\mathcal{J}}(A, B) = 1 - sim_{\mathcal{J}}(A, B)$

For the purposes of a similarity measure, discussed in Sect. 6.1, we require the similarity measure to be symmetric, i. e., the order in which the successor relations of two process models are compared shall not affect the similarity result. As a consequence, we need to compare relations of both process models that are of the same type. For now, we resort to the up-to-k-successor relation, $>_k^S$, where k is a parameter to control the sensitivity of the similarity measure. Following our discussion in Sect. 4.2, larger k suggest a lower sensitivity to behavioral differences. Contrarily, relations may incorporate more undesired relations due to cyclic structures. If, for instance, there is a global loop, as in a short-circuit net, each action will be in successorship with each other action in $>_{b(S)}^S$. We examine the practical influence of this parameter in Sect. 7.4.

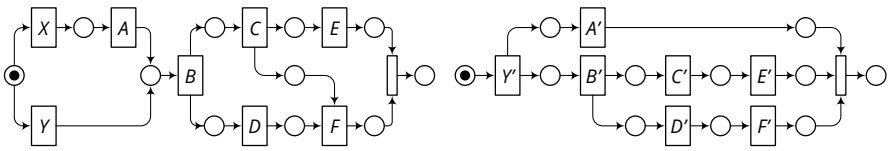
Intersection of successor relations

Similarity assessment based on successor relations requires an alignment of both process models, i. e., the discovery of corresponding actions. In Sect. 3.4 we provided an overview of techniques to construct such an alignment. As the sets of actions of two process models are distinct, we apply the intersection of behavioral relations, cf. Def. 4.3, to compute the number of common successor relations. Moreover, we reformulate the cardinality of the union of two sets by $|A \cup B| = |A| + |B| - |A \cap B|$.

Definition 6.2 (Naive Up-to-k-successor Relation Similarity)**.**
Let $S = (N, M_0)$ and $S' = (N', M'_0)$ be net systems with $N = (P, T, F)$ and $N' = (P', T', F')$, and $(\backsim, \phi)$ an elementary alignment of their sets of transitions, i. e., $\backsim \subseteq T' \times T$.
Let $>_k^S \subseteq T \times T$ and $>_k^{S'} \subseteq T' \times T'$ be the up-to-k-successor relations of systems S and S', respectively.
The *naive similarity* of $>_k^{S'}$ and $>_k^S$ is defined as

$$sim_{>}(>_k^{S'}, >_k^S) = \frac{\left|>_k^{S'} \widetilde{\cap} >_k^S\right|}{\left|>_k^{S'}\right| + \left|>_k^{S'}\right| - \left|>_k^{S'} \widetilde{\cap} >_k^S\right|} \quad \blacktriangleleft$$

For an example to compute the naive up-to-k-successor relation similarity, we refer to the process models depicted in Fig. 34 at the beginning of this chapter. Figure 35 shows the corresponding workflow systems of the two process models. The start events of the BPMN models have been translated to transitions in the workflow systems, whereas the mutually exclusive start events service notification and customer complaint of Fig. 34a have been captured as exclusive transitions with a common initial place in Fig. 35a. The actions and events of the process models are represented by their character annotations A–F, X, Y, and A′–F′, X′, respectively. The up-to-k-successor relations of these net systems, depicted in Tab. 7, are efficiently computed from the net systems as described in Sect. 4.2. In the matrices, we neglect transitions that have no business semantics, i. e., remained unlabeled in the net systems. Common 1-successor relation pairs, computed by the intersection of behavioral relations, cf. Def. 4.3, are highlighted dark gray in Tab. 7; common relation pairs that result from up-to-k-successor relations with k being the successor bound are shaded light and dark, as they include the 1-successors.



(a) Net system S of Fig. 34a. (b) Net system S' of Fig. 34b.

Figure 35: Net systems of example process models from Fig. 34.

From the set of common successor relation pairs, we compute the naive up-to-k-successor relation similarity.

$$sim_{>}(>_1^{S'}, >_1^{S}) = {}^{13}/_{25} \approx 0.52$$
$$sim_{>}(>_{b(S')}^{S'}, >_{b(S)}^{S}) = {}^{23}/_{36} \approx 0.64$$

The reason for the moderate similarities lies within transition A′ that is concurrent to, and therefore in weak order with, all transitions but Y′ in Fig. 35b, which is captured by up-to-k-successor relations for any $k \geq 1$. In Fig. 35a, the corresponding transition A is only in weak order with X.

The reason for the higher similarity among the weak order relation is due to the lower sensitivity of a far look-ahead that has already been emphasized in Sect. 4.2. Actions C–F and C′–F′ have different local dependencies in the workflow systems. However, with $k \geq 3$, these differences are mostly interpolated, which yields more commonalities among the successor relations.

Alignment confidence

A process alignment $(\backsim, \phi)$ is essential to compare process models and is typically based on labels or annotations of actions of the process models. As it is unlikely that the labels of two process models are identical [92], alignments are generally the result of a solution to the matching problem that strives to identify the best matches between the elements of two conceptual models [86], i. e., an action of one process model is aligned with the action of another model that is most likely to resemble the same concept. These aligned actions are captured as pairs of transitions in the correspondence relation $\backsim$; the likeliness of resembling the same concept is captured by the confidence function, ϕ, in the alignment, cf. Def. 3.13. Some transitions may not have a correspondence in an alignment, e. g., X in Fig. 35a.

A low confidence value indicates that two actions may not represent the same task or function [300, 105, 223]. For instance, in Fig. 34 the action order spare parts (B′) of process model Fig. 34b is aligned

	X	Y	A	B	C	D	E	F
X	·	·	1	2	3	3	4	4
Y	·	·	·	1	2	2	3	3
A	·	·	·	1	2	2	3	3
B	·	·	·	·	1	1	2	2
C	·	·	·	·	·	1	1	1
D	·	·	·	·	1	·	1	1
E	·	·	·	·	·	1	·	1
F	·	·	·	·	·	·	1	·

(a) $>_k^S$ of net system in Fig. 35a.

	Y′	A′	B′	C′	D′	E′	F′
Y′	·	1	1	2	2	3	3
A′	·	·	1	1	1	1	1
B′	·	1	·	1	1	2	2
C′	·	1	·	·	1	1	1
D′	·	1	·	1	·	1	1
E′	·	1	·	·	1	·	1
F′	·	1	·	1	·	1	·

(b) $>_k^{S'}$ of net system in Fig. 35b.

Table 7: Up-to-k-successor relations for example net systems of Fig. 34. Highlighted fields indicate the intersection of behavioral relations $>_k^{S'} \mathbin{\tilde{\cap}} >_k^S$ for $k \geq \max(b(S), b(S'))$ (gray) and for $k = 1$ (dark gray).

with order parts (B) of model Fig. 34a. However, in the second process it is not clear, which kind of parts is ordered, and therefore, these two actions may not denote identical concepts in the processes. The same holds also for other actions, e. g., manage stock Fig. 34a is less specific than update stock Fig. 34b and may address more wide-ranging tasks.

Despite the capability of an alignment to assess the similarity of actions, the majority of approaches to process model similarity resorts to correspondences without the confidence function for the computation of a behavioral distance or similarity value, cf. Sect. 6.2. In a similar fashion, Def. 6.2 only counts the number of corresponding behavioral relation pairs, but neglects the similarity between the aligned actions. To incorporate the confidence of an alignment into the similarity of process model behavior, we compute the aggregated confidence score of common successor relation pairs.

From Def. 4.3 follows that the intersection of behavioral relations consists of pairs of correspondences, $(\backsim_x, \backsim_y) \in R' \mathbin{\tilde{\cap}} R$, for which the confidence is provided by $\phi(\backsim_x)$ and $\phi(\backsim_y)$, respectively.

Definition 6.3 (Aggregated Confidence Score)**.**
Let $(\backsim, \phi)$ be an elementary alignment and $X \subseteq \backsim \times \backsim$ a binary relation that captures pairs of correspondences over the alignment.
The *aggregated confidence score* is a function $\Phi : \mathcal{P}(X) \mapsto \mathbb{R}$ that aggregates the confidence of correspondences over X such that

$$\Phi(X) = \sum_{(\backsim_x, \backsim_y) \in X} \tfrac{1}{2} \cdot \big(\phi(\backsim_x) + \phi(\backsim_y)\big) \qquad \blacktriangleleft$$

From the definition follows that $0 \leq \Phi(X) \leq |X|$, whereas $\Phi(X) = |X|$ holds, if and only if the confidence for each correspondence is 1. We incorporate the aggregated confidence score in the successor similarity as follows.

Definition 6.4 (Probabilistic Up-to-k-successor Relation Similarity)**.**
Let $S = (N, M_0)$ and $S' = (N', M'_0)$ be net systems with $N = (P, T, F)$ and $N' = (P', T', F')$, and $(\backsim, \phi)$ an elementary alignment of their sets of transitions, i. e., $\backsim \subseteq T' \times T$.
Let $>_k^S \subseteq T \times T$ and $>_k^{S'} \subseteq T' \times T'$ be up-to-k-successor relations of systems S and S', respectively.
The *probabilistic similarity* of $>_k^{S'}$ and $>_k^S$ is defined as

$$sim_\Phi(>_k^{S'}, >_k^S) = \frac{\Phi\big(>_k^{S'} \mathbin{\tilde{\cap}} >_k^S\big)}{\big|>_k^{S'}\big| + \big|>_k^{S'}\big| - \big|>_k^{S'} \mathbin{\tilde{\cap}} >_k^S\big|} \qquad \blacktriangleleft$$

The term *probabilistic* refers to the purpose of the alignment confidence in the similarity measure to assess the probability that a pair of correspondences in $>_k^{S'} \mathbin{\tilde{\cap}} >_k^S$ represents the same behavioral relation between tasks in the process models. Consequently, we compute this probability by the average of the confidence values of the involved pairs of correspondences.

	A′	B′	C′	D′	E′	F′	Y′
A	**1.000**	0.091	0.154	0.182	0.333	0.091	0.158
B	0.235	**0.647**	0.294	0.000	0.235	0.176	0.105
C	0.200	0.267	**0.667**	0.067	0.267	0.200	0.211
D	0.167	0.083	0.154	**0.333**	0.083	0.167	0.211
E	0.667	0.083	0.231	0.083	**0.667**	0.083	0.105
F	0.118	0.235	0.235	0.059	0.176	**0.471**	0.421
X	0.150	0.150	0.200	0.250	0.200	0.200	0.150
Y	0.167	0.278	0.222	0.222	0.167	0.278	**0.895**

Table 8: Normalized string edit similarity for the actions of example process models in Fig. 34.

Similarity of action labels

Table 8 shows similarity values computed for the labels of all pairs in the Cartesian product of actions of the example process models depicted in Fig. 34. Since the focus of this work is not on the construction of an alignment, but rather on the incorporation of alignment confidence into a process similarity measure, we resort to a simple and purely syntactical confidence function for labels.

Notation 6.1 (Normalized String Edit Similarity). Let Σ be an alphabet and $a, b \in \Sigma^*$ two words, and *ld* the string edit distance [168].

The *normalized string edit similarity* is defined as follows.

$$ldn(a,b) = 1 - \frac{ld(a,b)}{max(|a|,|b|)}$$ ◀

The normalized string edit similarity normalizes the string edit distance [168] by the size of the longer string and subtracts the result from 1. It returns values between 0 and 1, whereas 1 indicates that two labels are identical.

Bold figures in Tab. 8 identify the correspondence pairs that show the optimal matching of actions based on their similarity. These values are used as confidence values in the alignment. It is worth mentioning that $ldn(\mathsf{E}, \mathsf{A}') = ldn(\mathsf{E}, \mathsf{E}') \approx 0.667$, that is, also A′ could have been chosen as a corresponding action for E, if it was not aligned with A for its significantly higher similarity $ldn(\mathsf{A}, \mathsf{A}') = 1$. Likewise, X has no correspondence assigned.

From the confidence values, we can compute the aggregated confidence score and probabilistic up-to-*k*-successor similarity that yield the following values:

$$sim_\Phi(>_1^{S'}, >_1^{S}) \approx {}^{7.52}/_{25} \approx 0.30,$$
$$sim_\Phi(>_{b(S')}^{S'}, >_{b(S)}^{S}) \approx {}^{14.66}/_{36} \approx 0.41$$

Whereas these similarity values are considerably smaller than the above for the naive successor similarity, due to the aggregated confidence score, we argue that they represent the similarity of the process models more precisely. If we compared further models with above examples that show higher confidence in their alignments and equal

overlap of corresponding successor relations, they will also show a greater similarity, which is in line with the intuition that they were better matches.

6.4 A CONFIGURABLE SIMILARITY MEASURE

In a process model, pairs of actions can be related by the up-to-k-successor relation in different ways. The relation may hold in one direction, in both directions, or not at all. For instance, if a pair of transitions $(x, y) \in >_k^S$ but $(y, x) \notin >_k^S$, then y cannot be executed before x in any firing sequence of length $k+1$ of a process model. In particular, if $k \geq b(S)$ then there exists no trace at all, in which y can be executed before x. These more expressive behavioral relationships are captured by relation set.

Relation Sets

Relation sets distinguish whether a pair of actions is in strict order, interleaving order, or exclusive, and have been defined before for the precedence relation ($k = 1$) in the context of process mining [195] where they are called footprints [284]. Similarly, they have been defined for the weak order relation to measure consistency of process models on different levels of granularity [315, 319]. Their application over the generalized up-to-k-successor relation has first been presented in [314].

Definition 6.5 (k-Relation Set)**.**
Let $S = (N, M_0)$ be a net system with $N = (P, T, F)$ and $>_k^S \subseteq (T \times T)$ the up-to-k-successor relation of S. A pair of transitions $(x, y) \in >_k^S$ is in one of the following relations:

- exclusiveness, $+_k^S$, iff $(x, y) \notin >_k^S$ and $(y, x) \notin >_k^S$
- strict order, $\rightarrow_k^S$, iff $(x, y) \in >_k^S$ and $(y, x) \notin >_k^S$
- interleaving order, $||_k^S$, iff $(x, y) \in >_k^S$ and $(y, x) \in >_k^S$

$\mathcal{B}_k^S = \left\{ +_k^S,\ \rightarrow_k^S,\ ||_k^S \right\}$ is the k-relation set of S. We define $(x, y) \in \mathcal{B}_k^S$ iff $(x, y) \in \left(+_k^S \cup \rightarrow_k^S \cup ||_k^S \right)$. ◀

For a relation set, we also consider the reverse strict order relation, denoted by $\leftarrow_k^S$. It is defined as the inverse relation of strict order $\rightarrow_k^S$, i. e., $(x, y) \in \rightarrow_k^S \iff (y, x) \in \leftarrow_k^S$. If it is clear from the context, we omit the identifier of the net system for relation sets, hereafter, e. g., $+_k$ denotes the exclusiveness relation of a k-relation set $\mathcal{B}_k$.

The definition of a relation set induces various properties for the comprised behavioral relations, see [319, 284, 314] for proofs of these properties. Relations $+_k^S$ and $||_k^S$ are symmetric; relations $\rightarrow_k^S$ and $\leftarrow_k^S$ are antisymmetric and irreflexive. Also, the four relations $\rightarrow_k^S$,

$\leftarrow_k^S$, $||_k^S$, and $+_k^S$ are mutually exclusive and partition the Cartesian product of actions of a process model, i. e., each pair of observable actions is either in strict or inverse strict order, interleaving order, or exclusive.

Look-ahead parameter k

The influence of the look-ahead parameter k on the identification of commonalities has been elaborated on in Sect. 4.2, which also applies to k-relation sets. Consider a cyclic model that can produce arbitrary many repetitions of a firing sequence $\langle A, B, C\rangle$. In $\mathcal{B}_1$ it holds $\rightarrow_1 = \{(A,B),(B,C),(C,A)\}$ and $(A,C),(B,A) \in +_1$, whereas in $\mathcal{B}_2$ it holds $(A,C),(B,A) \in \rightarrow_2$. For $k = 3$, the loop can be closed and any action is observed as a successor of any other action, i. e., all pairs of actions are in $||_3$. This implies that, while increasing k adds information about successorship of actions in process models, it also hides certain information, e. g., the execution order of actions that are part of a cycle may be lost or iterations and concurrent enabling of actions may not be distinguished anymore [319].

Since relation sets partition the Cartesian product of actions, we cannot represent the k-relation set over increasing k in one matrix for visualization purposes. Table 9 and Tab. 10 visualize the k-relation sets for the exemplary business processes, where $+$ denotes a pair of actions is exclusive, $\rightarrow$ denotes a pair of actions is in strict order, and $||$ denotes a pair of actions is in interleaving order. Shaded fields

	X	Y	A	B	C	D	E	F
X	+	+	→	+	+	+	+	+
Y	+	+	+	→	+	+	+	+
A	←	+	+	→	+	+	+	+
B	+	←	←	+	→	→	+	+
C	+	+	+	←	+	||	→	→
D	+	+	+	←	||	+	||	→
E	+	+	+	+	←	||	+	||
F	+	+	+	+	←	←	||	+

(a) $\mathcal{B}_1^S$ of net system in Fig. 35a

	Y′	A′	B′	C′	D′	E′	F′
Y′	+	→	→	+	+	+	+
A′	←	+	||	||	||	||	||
B′	←	||	+	→	→	+	+
C′	+	||	←	+	||	→	||
D′	+	||	←	||	+	||	→
E′	+	||	+	←	||	+	||
F′	+	||	+	||	←	||	+

(b) $\mathcal{B}_1^{S'}$ of net system in Fig. 35b

Table 9: k-relation set for example processes, $k = 1$.

	X	Y	A	B	C	D	E	F
X	+	+	→	→	→	→	→	→
Y	+	+	+	→	→	→	→	→
A	←	+	+	→	→	→	→	→
B	←	←	←	+	→	→	→	→
C	←	←	←	←	+	||	→	→
D	←	←	←	←	||	+	||	→
E	←	←	←	←	←	||	+	||
F	←	←	←	←	←	←	||	+

(a) $\mathcal{B}^S_{b(S)}$ of net system in Fig. 35a

	Y′	A′	B′	C′	D′	E′	F′
Y′	+	→	→	→	→	→	→
A′	←	+	||	||	||	||	||
B′	←	||	+	→	→	→	→
C′	←	||	←	+	||	→	||
D′	←	||	←	||	+	||	→
E′	←	||	←	←	||	+	||
F′	←	||	←	||	←	||	+

(b) $\mathcal{B}^{S'}_{b(S')}$ of net system in Fig. 35b

Table 10: k-relation set for example processes, $k \geq \max(b(S), b(S'))$.

indicate common relation pairs of the same elementary type for the example process models, computed by the behavioral relation intersection, cf. Def. 4.3.

Strictness of relation types

The different relation types of a k-relation set can be classified according to a notion of strictness, based on the imposed effect on firing sequences of a process model [320]. As such, exclusiveness can be seen as the strictest imposition, as it ensures that two actions cannot occur together in any sequence of length k. For k being greater than or equal to the successor bound $b(S)$, this coincides with the mutual exclusion of actions in the behavior of the process model. For smaller k, in particular $k = 1$, this exclusiveness denotes the absence of co-occurrence of two actions in close proximity in a trace. Strict order and inverse strict order are less restricting as they denote that pairs of actions may co-occur in a trace. If they occur together in a firing sequence they show always the same ordering. We argue that strict order is more restricting the smaller the look-ahead k is, as this requires a certain proximity of the actions in a trace. In contrast, for large k this only requires the co-occurrence of a pair of actions in the same order. The weakest restriction is certainly imposed by interleaving order, which denotes that a pair of actions may occur together in a trace in any order. Recall that it is not required that both execution orders are satisfied within one process instance. Again, increasing k weakens this restriction.

Relation Set Similarity

Each of the elementary relation types of relation sets addresses particular aspects of the behavior of a process model, which we consider in the definition of elementary similarities.

Definition 6.6 (Elementary Relation Set Similarities)**.**
Let $S = (N, M_0)$ and $S' = (N', M_0')$ be net systems with $N = (P, T, F)$ and $N' = (P', T', F')$, and $(\backsim, \phi)$ an elementary alignment of their sets of transitions, i. e., $\backsim \subseteq T' \times T$.
Let $\mathcal{B}_k^S$ and $\mathcal{B}_k^{S'}$ be the k-relation sets of systems S and S', respectively. $\mathcal{B}$ denotes the universe of all possible relation sets.
We define the following elementary relation set similarities $sim_i : \mathcal{B} \times \mathcal{B} \mapsto [0, 1]$, $i \in \{+, \rightarrow, ||\}$.

Exclusiveness Similarity
$$sim_+(\mathcal{B}_k^{S'}, \mathcal{B}_k^S) = \frac{\Phi\left(+_k^{S'} \mathbin{\tilde{\cap}} +_k^S\right)}{\left|+_k^{S'}\right| + \left|+_k^S\right| - \left|+_k^{S'} \mathbin{\tilde{\cap}} +_k^S\right|}$$

Strict Order Similarity
$$sim_\rightarrow(\mathcal{B}_k^{S'}, \mathcal{B}_k^S) = \frac{\Phi\left(\rightarrow_k^{S'} \mathbin{\tilde{\cap}} \rightarrow_k^S\right)}{\left|\rightarrow_k^{S'}\right| + \left|\rightarrow_k^S\right| - \left|\rightarrow_k^{S'} \mathbin{\tilde{\cap}} \rightarrow_k^S\right|}$$

Interleaving Order Similarity
$$sim_{||}(\mathcal{B}_k^{S'}, \mathcal{B}_k^S) = \frac{\Phi\left(||_k^{S'} \mathbin{\tilde{\cap}} ||_k^S\right)}{\left||_k^{S'}\right| + \left||_k^S\right| - \left||_k^{S'} \mathbin{\tilde{\cap}} ||_k^S\right|}$$ ◀

We discuss the characteristics of these elementary similarities with regard to the influence of the look-ahead parameter k, before we show how they can be combined in a configurable similarity function.

Exclusiveness similarity

A pair of transitions is in the exclusiveness relation of a k-relation set, if the transitions cannot co-occur in any subsequence of a trace of length $k+1$. As discussed above, increasing k will reveal new successors of an action and therefore, eliminate pairs from the exclusiveness relation. Conversely, if two actions are exclusive in $+_k$ they are also exclusive in $+_{k-1}$. Table 9 shows that, in particular, 1-relation sets show a large number of exclusiveness relations due to the small look-ahead. From the aggregated confidence score over the common exclusiveness relations of S' and S, we compute $sim_{+}(\mathcal{B}_1^{S'}, \mathcal{B}_1^{S}) \approx {}^{12.83}/_{41} \approx 0.31$. The additional transition X leads to a large number of unmatched exclusiveness relations, and therefore to a lower similarity. The exclusiveness relation with successor bound yields $sim_{+}(\mathcal{B}_{b(S')}^{S'}, \mathcal{B}_{b(S)}^{S}) \approx {}^{4.68}/_{12} \approx 0.39$.

Strict order similarity

The execution order of actions that remains invariant in all instances of a business process model that execute these actions is reflected by the strict order relation $\rightarrow_k$. Hence, the strict order similarity rewards commonalities of strict order relations. The strict order similarity considers only $\rightarrow_k$, because from above notion of $\leftarrow_k$ follows that for any pair of actions (x, y) that is in the reverse strict order, the inverted pair (y, x) is in the strict order relation. From the computation of the strict order similarity of the example business processes, we get $sim_{\rightarrow}(\mathcal{B}_1^{S'}, \mathcal{B}_1^{S}) \approx {}^{2.99}/_{9} \approx 0.33$ and $sim_{\rightarrow}(\mathcal{B}_{b(S')}^{S'}, \mathcal{B}_{b(S)}^{S}) \approx {}^{7.06}/_{24} \approx 0.29$.

The look-ahead parameter k has a noteworthy influence on action pairs of the strict order relation, because their successorship may not be discovered in a firing sequence with k-look-ahead, but with $k+1$. Increasing k further may disclose also the inverse execution order of that pair of actions and yield the interleaving order relationship.

Interleaving order similarity

Interleaving order is the weakest imposition on the relations between two actions, as it only states that they can be executed in any order in one process instance. Thus, the interleaving order similarity also rewards matching pairs, if they are, e. g., executed in parallel in one process model and as part of a control flow cycle in another. Nevertheless, iterations and concurrency can be distinguished up to certain limitations, cf. Sect. 4.2. As a result from the broader scope of a relation due to a far look-ahead, the interleaving order relationship shows inverse behavior over an increasing k compared to the exclusiveness relation, i. e., if a pair of actions is in $||_k$ it is also in $||_{k+1}$. Since the interleaving order relations are equal for all k in the given examples respectively, their relation sets have an equal interleaving order similarity, $sim_{||}(\mathcal{B}_1^{S'}, \mathcal{B}_1^{S}) = sim_{||}(\mathcal{B}_{b(S')}^{S'}, \mathcal{B}_{b(S)}^{S}) \approx {}^{3.14}/_{18} \approx 0.17$.

Based on the elementary similarity measures discussed above, the relation set similarity is constructed as follows.

Definition 6.7 (k-Relation Set Similarity & Distance)**.**
Let $S = (N, M_0)$ and $S' = (N', M_0')$ be net systems with $N = (P, T, F)$ and $N' = (P', T', F')$, and $(\backsim, \phi)$ an elementary alignment of their sets of transitions, i. e., $\backsim \subseteq T' \times T$.
Let $\mathcal{B}_k^S$ and $\mathcal{B}_k^{S'}$ the k-relation sets of systems S and S', respectively. $\mathcal{B}$ denotes the universe of all possible behavioral relation sets.
The *relation set similarity*, $sim_{\mathcal{B}} : \mathcal{B} \times \mathcal{B} \mapsto [0, 1]$, is composed of the elementary similarities

$$sim_{\mathcal{B}}(\mathcal{B}_k^{S'}, \mathcal{B}_k^S) = \sum_{i \in \{+, \rightarrow, ||\}} w_i \cdot sim_i(\mathcal{B}_k^{S'}, \mathcal{B}_k^S)$$

with the weighting factors $w_i \in \mathbb{R}$, $0 \leq w_i \leq 1$ such that it holds $\sum_{i \in \{+, \rightarrow, ||\}} w_i = 1$.
The *relation set distance* is defined by

$$d_{\mathcal{B}}(\mathcal{B}_k^{S'}, \mathcal{B}_k^S) = 1 - sim_{\mathcal{B}}(\mathcal{B}_k^{S'}, \mathcal{B}_k^S). \quad \blacktriangleleft$$

The k-relation set similarity aggregates the elementary similarities, where each elementary similarity is assigned a weight to account for the respective similarity's impact on the overall similarity. Under the assumption that all weights are equal, we arrive at the following relation set similarity values for the workflow systems shown in Fig. 35.

$$sim_{\mathcal{B}}(\mathcal{B}_1^{S'}, \mathcal{B}_1^S) \approx 0.27$$
$$sim_{\mathcal{B}}(\mathcal{B}_{b(S')}^{S'}, \mathcal{B}_{b(S)}^S) \approx 0.29$$

These similarity values are generally smaller than those for the probabilistic similarity of up-to-k-successor relations, cf. Sect. 6.3. This is caused by the explicit distinction of relation types in contrast to the successor relation, where a pair of actions that is only in one order relation. For instance in Tab. 7, $(\mathsf{A}, \mathsf{B}) \in >_k$ but $(\mathsf{B}, \mathsf{A}) \notin >_k^S$, is at least partially matched with a corresponding pair of actions that is in order and inverse order relation, i. e., $(\mathsf{A}', \mathsf{B}'), (\mathsf{B}', \mathsf{A}') \in >_k^{S'}$. In contrast, this pair of actions does not match in their relation sets, cf. Tab. 9 and Tab. 10.

Configuration of the Relation Set Similarity

The k-relation set similarity and distance function introduced above, combine a set of elementary similarities, which need to be assigned a weight in the aggregated similarity. Choosing these weights allows for the configuration of the similarity for a dedicated collection of process models, which enables emphasizing particular aspects of process models' behavior for search. However, it also raises the question, how these weights shall be set in the first place.

If a training set of process models along with human similarity assessment is available, the optimal weights may be determined experimentally. That is, when comparing various configurations of the computed similarity with the human assessment, the best configuration shows the smallest deviation of computed values and human

judgement over the complete training set. Unfortunately, generating a human assessment of the similarity of reference process models is a laborious and time-consuming task, and often not feasible in an organization. Therefore, we outline how the weights may be determined from the characteristics of a process model collection, or, more precisely from a set of k-relation sets.

Essentially, our approach leverages the sizes of the relations that are considered by the elementary similarities. The intuition is that the more frequent a certain behavioral relation is observed in a corpus of relation sets, the more importance shall be assigned to the similarity that is grounded on this relation. Hence, we define the weights as follows.

Definition 6.8 (Weights for k-Relation Set Similarity & Distance)**.** Let $\mathcal{B} = \left\{\mathcal{B}_k^{S_1}, ..., \mathcal{B}_k^{S_n}\right\}$ be a set of k-relation sets with $\mathcal{B}_k^{S_i} = \left\{+_k^{S_i}, \rightarrow_k^{S_i}, ||_k^{S_i}\right\}$ for $1 \leq i \leq n$.
We define the size l_i of one of these sets as $l_i = \left|+_k^{S_i} \cup \rightarrow_k^{S_i} \cup \leftarrow_k^{S_i} \cup ||_k^{S_i}\right|$. Then, the weights w_+, $w_\rightarrow$, and $w_{||}$ for the respective elementary similarities induced by $\mathcal{B}$ are defined as follows.

$$w_+(\mathcal{B}) = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{\left|+_k^{S_i}\right|}{l_i} \quad w_\rightarrow(\mathcal{B}) = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{\left|\rightarrow_k^{S_i} \cup \leftarrow_k^{S_i}\right|}{l_i} \quad w_{||}(\mathcal{B}) = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{\left||_k^{S_i}\right|}{l_i} \quad \blacktriangleleft$$

An action x is exclusive with itself, i. e., $(x, x) \in +_k^S$, if it can appear at most once in a trace of a system S. Otherwise, e. g., if it is part of a loop, it is in interleaving order with itself, i. e., $(x, x) \in ||_k^S$. Relation sets show a particular property, in that, they are mirrored along the diagonal of self-relations, which follows from $||_k^S$ and $+_k^S$ being symmetric and $\rightarrow_k^S$ being the inverse relation of $\leftarrow_k^S$. Hence, in order to treat all relations pairs without a bias, the weight for the strict order similarity must acknowledge the number of strict order and inverse strict order relation pairs.

The definition of these weights satisfies the requirements imposed by Def. 6.7. The weights indeed sum up to 1, which follows from the fact that the relations of each relation set are mutually exclusive and partition the Cartesian product of actions over which they are defined. In Sect. 7.4, we evaluate the effectiveness of determining these weights from the frequency of the according relations in a process model collection.

6.5 EFFICIENT SIMILARITY SEARCH

One particular challenge of similarity search is the efficient discovery of those process models in a repository that satisfy the query, where efficient denotes that the query must not be compared against each model stored in a process model repository [276]. This issue becomes apparent in related work on process model similarity, where

efficiency is rarely addressed at all, cf. Sect. 6.2. This can only be solved by means of algorithms and data structures that avoid exhaustive search, i. e., indexes. Unfortunately, distance based search cannot be solved by means of traditional indexing, e. g., binary tree indexes, as there exists no ordering among the process models, but only a pairwise distance. Also, hashing cannot be applied, as it separates data points into clusters, which does not align well with the continuity of the similarity notion.

In general, we observed two approaches to improve the efficiency of similarity search. On the one hand, it is possible to decompose the search into two stages; a computationally inexpensive filter function excludes process models that cannot satisfy the query and provides a list of candidates which is then exhaustively examined by means of the actual distance function. The *filter and verification* paradigm has been presented in Sect. 5.5, but is suited for similarity search only to a limited extent. Candidates that show only low similarity with a query cannot be excluded if there exist not sufficiently many matches with higher similarity, i. e., candidates can be excluded only, if they show no commonalities at all with a query. On the other hand, specific properties of the distance function allow partitioning the space spanned by process models and excluding partitions during search. This is the case, if process models and distance function form a metric space [143].

A proper metric

Metric spaces are a generalization of vector spaces, where nothing but a pairwise distance can be computed between the objects contained in that space and the distance is a metric.

Definition 6.9 (Metric Space)**.**
A *metric* is a distance function $d : \mathcal{D} \times \mathcal{D} \mapsto \mathbb{R}$ between objects of domain $\mathcal{D}$ with the following properties:

- Symmetry: $\forall\, o_i, o_j \in \mathcal{D} : d(o_i, o_j) = d(o_j, o_i)$
- Non-negativity: $\forall\, o_i, o_j \in \mathcal{D} : d(o_i, o_j) \geq 0$
- Identity: $\forall\, o_i, o_j \in \mathcal{D} : d(o_i, o_j) = 0 \iff o_i = o_j$
- Triangle Inequality: $\forall\, o_i, o_j, o_k \in \mathcal{D} : d(o_i, o_k) \leq d(o_i, o_j) + d(o_j, o_k)$

A *metric space* is a pair $(\mathcal{D}, d)$. ◂

Based on metric spaces, a variety of index structures and algorithms has been devised that aim at partitioning the space and excluding some of these partitions during search if none of the contained objects in a partition can satisfy a query due to their distance. With that regard, the *triangle inequality* property makes efficient search in metric spaces possible, because it allows reducing the number of comparison operations required for search through a notion of transitivity [276]: If the distances between two pairs of three data points o_i, o_j, o_k are given, the remaining distance can be determined by its

lower and upper boundary, i. e., $|d(o_i, o_j) - d(o_j, o_k)| \leq d(o_i, o_k) \leq d(o_i, o_j) + d(o_j, o_k)$.

With the aforementioned transformation of the Jaccard Coefficient into a metric [174], the *naive up-to-k-successor relation similarity*, cf. Def. 6.2, can be transformed into a distance that is a proper metric: $d_>(>_k^{s'}, >_k^s) = 1 - sim_>(>_k^{s'}, >_k^s)$ and can be used to efficiently search in a metric space. Before we present the application of the *relation set distance*, cf. Def. 6.7, to metric space search, we elaborate on the background of metric space indexes in general, and how an index is constructed and enables efficient search in particular.

Indexing Techniques for Similarity Search

Existing approaches to metric space indexing expose a great diversity in the way they organize data and search for relevant results of a query, cf. [47, 122, 333]. However, most of them are based on the same principals. With the help of a metric, a data set is partitioned into several subsets of data points that are more similar to each other than others. Upon search, only subsets that are relevant to the query must be examined exhaustively to provide a ranking. Irrelevant subsets can be safely excluded from search. Recursive partitioning of the subsets increases efficiency.

Literature on similarity search [277, 47, 333] distinguishes two types of indexing techniques according to the way they partition the data set: *ball decomposition* and *hyperplane decomposition*. The major difference is, how many *pivots* are used to partition the data set. A pivot $p \in \mathcal{D}$ is a reference point in the data set. We discuss these types briefly, whereas more detailed discussions can be found in [47, 122, 333].

Ball decomposition

Techniques that resort to ball decomposition use only one pivot p and a distance radius $r(p)$ that splits the space in two partitions. The name of this approach yields from the intuitive understanding of a sphere around p that disseminates the space, cf. Fig. 36a. All data points o_i that are closer to p than $r(p)$, i. e., $d(p, o_i) \leq r(p)$,

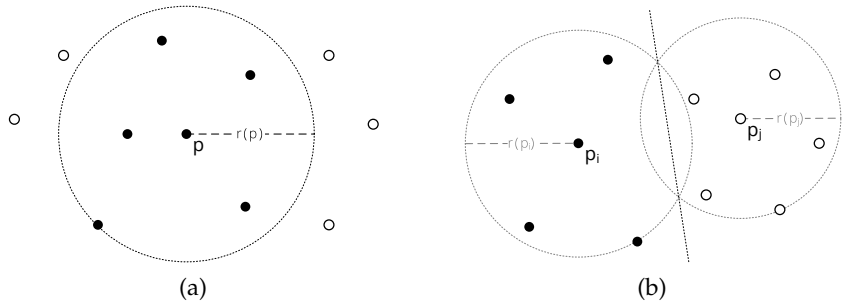


Figure 36: Examples for ball decomposition (a) and hyperplane decomposition (b).

are within the first partition (black data points), all other objects in the second partition (white data points). This can be extended by several consecutive radii, which create rings around p that contain data points. During search, only rings that are close enough to the query are considered for further examination. Examples for ball decomposition are the Vantage Point Tree [277, 332] that uses only one radius r, maintained as the exact median of the set of all data points. Thus, each subset contains equally many data points. The Burkhard-Keller Tree [44] — one of the earliest indexing approaches for metric spaces — relies on a distance function that yields few discrete values which result in a classification of data points. During search, the distance between pivot and query is used to identify a class that is then searched exhaustively.

Hyperplane decomposition

In contrast, hyperplane decomposition uses more than one pivot, and for each pivot a partition is created. A data point o_i is assigned to the partition of the pivot p_j it is most similar to. This approach can be visualized through a plane that divides the space between two pivots, cf. Fig. 36b, hence its name. If, during search, a search query is far enough from the partitioning hyperplane, i. e., further than the maximum accepted distance of the query, $r(q)$, the partition opposite of that plane can be safely excluded from search.

The Bisector Tree [140] computes covering spheres centered in the respective pivot containing all data points of a partition, visualized in Fig. 36b whose radius is determined by the most distant data point within a partition. While theses spheres may overlap, each datapoint belongs only to exactly one partition. If a search query is too far from this sphere, the partition can safely be pruned from search. The Geometric Near-neighbor Access Tree [37] and the M-Tree [52] extend this by using more pivots for a more diverse partitioning of the data set.

Efficient Search in the Absence of Coordinates

For our discussion of metric tree indexing, we resort to the *M-Tree* [52], a dynamic and balanced index that falls into the hyperplane decomposition class and uses several pivots to partition the data set. Dynamic means that it does not need to be rebuilt after any of the indexed data points changed but can be updated gradually. An index is balanced if all branches and subbranches have the same depth and fanout. This balances search times among different queries.

The M-Tree uses a set of pivots that partition the space of data points, illustrated in Fig. 37a. These pivots comprise a node of an M-Tree, cf. Fig. 37b: The root node n_{root} contains the three pivots p_1, p_2, and p_3. Each pivot p_j of a node n roots a subtree, which is denoted by $T(p_j)$ and referred to as *covering tree*. The covering tree of a pivot p_j contains all data points that are closer to p_j than to any other pivot of the same node, i. e., $\forall\, p_k, p_j \in n \wedge p_k \neq p_j\ \forall\, o_i \in T(p_j)$:

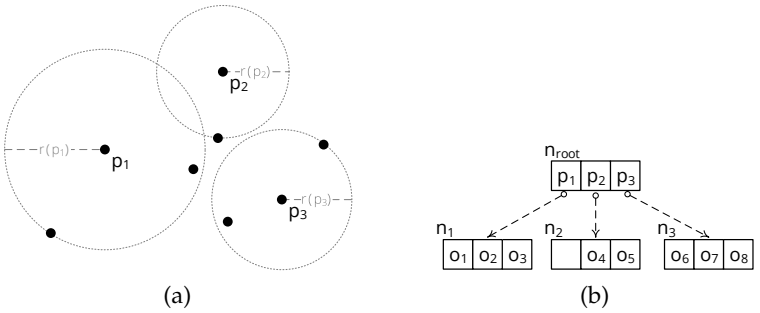


Figure 37: Example of an M-Tree index: space partitioning (a) and index data structure (b).

$d(p_j, o_i) \leq \min(d(p_k, o_i))$. For instance, n_1 forms the covering tree $T(p_1)$ in Fig. 37b. In the M-Tree, pivots are chosen from the set of data points and thus, every pivot is also contained as data point in its covering tree. This is why, for instance, n_1 in Fig. 37b contains three data points. Leaf nodes contain all indexed data points.

Index construction

The number of pivots per node is bounded through upper and lower limits that keep the index balanced. To insert new data points o_n into the index, first the parent pivot p_p of a leaf node n is found that has the least distance to o_n (see below). The new data point is then inserted into n. If this exceeds the node's size, it will be split: Two new pivots p_{p1}, p_{p2} are chosen from $T(p_p)$ and its elements are distributed among two new subtrees $T(p_{p1})$ and $T(p_{p2})$. p_{p1} and p_{p2} replace p_p. If this exceeds the size of the node that contained p_p, also this node will be split, etc. This may propagate toward the root of the tree and create a new root node. This procedure keeps the index tree balanced at any time without the need to rebuild it entirely. Further details of dynamically manipulating an M-Tree index can be found in [52].

Every data point $o_i \in T(p_j)$ stores the distance to its parent pivot and each pivot maintains its covering radius $r(p_j)$, such that for all $o_i \in T(p_j)$ it holds $d(o_i, p_j) \leq r(p_j)$. The covering radius also outranges the distance of p_j to any data point in any of its subtrees on lower levels, which provides a strict partitioning of the set of data points.

Range query

Similarity search requires a query model $q \in \mathcal{D}$ and a search radius $r(q) \in \mathbb{R}$ that defines how similar to q desired data points shall be, cf. Def. 6.1. Search starts in the root node: For each pivot $p_j \in n_{root}$, its distance $d(p_j, q)$ to the query is computed. The triangle inequality enables efficient search by safely pruning covering trees from further examination, i. e., if the distance $d(p_j, q)$ is larger than the sum of $r(p)$ and $r(q)$, i. e., $r(q) + r(p_i) < d(p_i, q)$, then no data point $o_i \in T(p_j)$ can exist within distance $r(q)$ to the query model, cf. Fig. 38a. In case $r(q) - r(p_i) \geq d(p_i, q)$, $T(p_i)$ is completely encompassed by the sphere around q, cf. Fig. 38b, and every data point $o_i \in T(p_j)$ matches

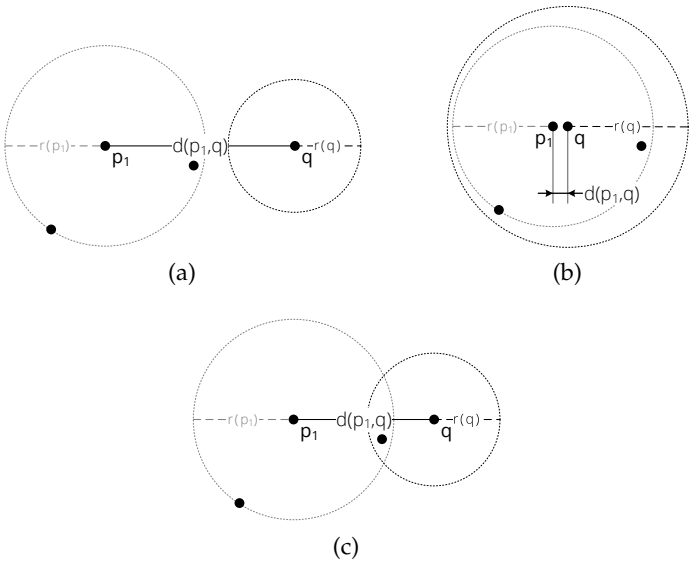


Figure 38: Search scenarios with covering tree $T(p)$ and similarity query q and search radius $r(q)$. (a) Exclusion, (b) Inclusion, (c) Intersection.

the desired search request. Thus, further examination of $T(p_j)$ is only required, if a ranking of search results is needed. Otherwise, i. e., if $r(q) + r(p_i) \geq d(p_i, q) > r(q) - r(p_j)$, the spheres of p_i and q intersect, illustrated in Fig. 38c. It is impossible at this step to determine, whether data points $o_i \in T(p_j)$ lie within this intersection and, if so, which, due to the lack of coordinates in metric spaces. Consequently, $T(p_j)$ must be further examined.

If $T(p_j)$ is not a leaf node, the same procedure to decide which of its subtrees can be pruned is applied. Otherwise, it can be decided without further distance computation, which of the data point $o_i \in T(p_j)$ match the query, since the distance $d(o_i, p)$ has been calculated during index construction: If $d(p_j, q) > r(q) + d(p_j, o_i)$ then o_i is further away from q than the acceptable distance $r(q)$. Since pivots are chosen from the set of data points to be indexed, computation of further distances can be avoided, by storing and recalling the distance of a query and pivots during search. Computation of the distance between query and matches is required for ranking.

Nearest neighbor query

For range query searches, $r(q)$ is provided with the query. However, this type of query is of limited use. On the one hand, the scale of the distance function is not necessarily aligned well with the expectation of a user. For instance, a value of 0.5 of a normalized distance function $d = 1/n$ with n being the number of differences between two objects does not express that 50% of the features of these two objects are shared. Nevertheless, such a distance may provide a useful ranking. On the other hand, there may exist only results with a low similarity,

see above examples for $sim_{\mathcal{B}}$, and choosing the query radius too high would not yield any results, even if meaningful matches are present.

These issues make it difficult to determine a useful search radius upfront. As a solution, nearest neighbor searches retrieve the n least distant data points for a given query. This is implemented as an extension to range queries, utilizing a result queue Ω of length n and a dynamic search radius $r(q)$. At the beginning of search, $r(q) = \infty$ and all pivots that have a smaller distance are added to the result queue in ascending order of their distance to the query, i. e., let i, j be indices that denote the position in Ω then $\forall\, i, j \in \mathbb{N}, 1 \leq i < j \leq |\Omega| : d(q, p_i) \leq d(q, p_j)$. Once, the result queue contains n data points, $r(q)$ will be reduced to the distance of the most distant data point in the queue, $d(q, p_n)$. As a result, data points that have a greater distance to q than the n^{th} element of the queue, will be pruned in the remainder of the search. If data points with a distance smaller than $d(q, p_n)$ are found, they will be inserted in the queue at a position according to their distance to q, and the search radius $r(q)$ is reduced to the updated n^{th} element of Ω. The iterative reduction of $r(q)$ ensures that no relevant models are excluded during search.

Searching in Metric Spaces with the Probabilistic Relation Set Distance

Efficient similarity search with a metric space index generally requires a similarity, more precisely its corresponding distance measure, to be a proper metric, i. e., satisfy the properties symmetry, non-negativity, identity, and triangle inequality. The triangle inequality cannot be guaranteed for the similarity measures presented in the previous sections, due to the incorporation of the alignment confidence into these measures.

Nevertheless, a distance d can be used for search in metric spaces, if there exists another distance d' that is a metric and it holds for all data points $o_i, o_j \in \mathcal{D}$ that $d'(o_i, o_j) \leq d(o_i, o_j)$, i. e., the metric underestimates the actual distance. Then, range query search can be conducted with the metric d', in that, the search result is verified and ranked with d. For nearest neighbor queries, d is used to manage the result queue. For this purpose, we introduce an auxiliary alignment that is a proper metric and show, how similarity search in metric spaces can be conducted by a combination of the auxiliary alignment and the alignment using actual confidence values.

Definition 6.10 (Auxiliary Alignment)**.**
Let $\{N_1, ...N_n\}$ with $N_i = (P_i, T_i, F_i)$ be a set of nets and $\{\backsim_1, ..., \backsim_{n-1}\}$ a set of elementary alignments over pairs of nets, i. e., $\backsim_i \subseteq T_i \times T_{i+1}$. $\backsim^*$ is the irreflexive transitive closure over $\bigcup_{1 \leq i < n} \backsim_i$ and ϕ' is a binary confidence function defined as

$$\phi'(x,y) = \begin{cases} 1 & \text{if } (x,y) \in \backsim^* \\ 0 & \text{else} \end{cases}$$

The *auxiliary alignment* is defined as $(\phi', \backsim^*)$ ◂

Let the elementary relation set similarities, cf. Def. 6.6, use $(\backsim^*, \phi')$ for an alignment, and $d'_i(\mathcal{B}_k^{S'}, \mathcal{B}_k^S) = 1 - sim_i(\mathcal{B}_k^{S'}, \mathcal{B}_k^S)$, $i \in \{+, \rightarrow, ||\}$ be elementary distance functions. From the transitivity of $\backsim^*$ follows that distance functions, d'_+, $d'_\rightarrow$, and $d'_{||}$, are metrics themselves derived from the Jaccard coefficient [174]. Then, their weighted sum, i. e., the k-relation set distance $d'_\mathcal{B}$, cf. Def. 6.7, is also a metric.

Lemma 6.1.
Let $x, y \in \mathcal{D}$. The weighted sum, $D(x,y) = \sum_i w_i \cdot d_i(x,y)$ of distance functions d_i is a metric, if for all $i \in \{1, ..., n\}$ it holds that $w_i \in \mathbb{R} \wedge 0 < w_i < 1$ and $d_i : \mathcal{D} \times \mathcal{D} \mapsto \mathbb{R}$ is a metric.

Proof. $D(x,y)$ holds the properties symmetry, non-negativity, identity, and triangle inequality.

Symmetry: From $d_i(x,y) = d_i(y,x)$ and the commutativity of the sum operation follows that $D(x,y) = D(y,x)$.

Non-negativity: From $d_i(x,y) \geq 0$ and $w_i > 0$ it follows directly that $w_i \cdot d_i(x,y) \geq 0$ and thus their sum $D(x,y) \geq 0$

Identity: From $d_i(x,y) = 0$ for all $i \in \{1, ..., n\}$ follows that $D(x,y) = 0$ and if $D(x,y) = 0$ all $d_i(x,y) = 0$ because $w_i \cdot d_i(x,y) \geq 0$ with $w_i > 0$. From $d_i(x,y) = 0$ however, it follows that $x = y$ and thus $D(x,y) = 0 \iff x = y$.

Triangle Inequality: From $d_i(x,z) \leq d_i(x,y) + d_i(y,z)$ and $w_i < 0$ follows that $w_i \cdot d_i(x,z) \leq w_i \cdot d_i(x,y) + w_i \cdot d_i(y,z)$ and through summation $D(x,z) \leq D(x,y) + D(y,z)$ □

Please note that behavioral distances d'_+, $d'_\rightarrow$, $d'_{||}$, and $d'_\mathcal{B}$ are metrics over relation sets, not process models. Relation sets are an abstraction of the behavior of a process models and establish a partition of all process models, i. e., $\pi(\mathcal{S}) = \{\pi_1, ..., \pi_n\}$, where process models with identical relation sets form an equivalence class $\pi_i = [S]_{\mathcal{B}_k}$, cf. Def. 4.2. That is, $\forall\, S, S' \in [S]_{\mathcal{B}_k} : \mathcal{B}_k^S = \mathcal{B}_k^{S'}$. Hence, we can apply the relation set distance on the equivalence classes of process models, i. e., $d_{\pi(\mathcal{S})}([S]_\mathcal{B}, [S']_\mathcal{B}) = d'_\mathcal{B}(\mathcal{B}_k^S, \mathcal{B}_k^{S'})$. Then, the identity property of

the metric distance holds for equivalence classes rather than process models. We argue that this abstraction is generally not an issue, as it is unlikely for several process models to have equal relation sets. This requires that they have the same set of observable actions in identical behavioral relations. In that case, these process models would indeed be very similar.

Searching with a probabilistic distance

With $d'_{\mathcal{B}}$ being a metric, an M-Tree index can be constructed where $d'_{\mathcal{B}}$ is used for the computation of the radii for the covering trees of pivots and of the distance of nodes to their parent pivot, see above. However, using $d'_{\mathcal{B}}$ for search neglects the confidence of aligning the query with candidates, which would yield false matches in view of the above definitions of behavioral similarity. On the other hand, $d_{\mathcal{B}}$ cannot be used for search either, as it may exclude partitions during search that contain relevant matches. From Def. 6.7 follows that, given an auxiliary alignment $(\backsim^*, \phi')$, for all pairs of correspondences $\backsim_x, \backsim_y \in \backsim^*$ holds $\Phi(\backsim_x, \backsim_y) \leq \Phi'(\backsim_x, \backsim_y)$ and therefore, for all $\mathcal{B}_k^{S'}, \mathcal{B}_k^{S} \in \mathcal{B}$ holds $d_{\mathcal{B}}(\mathcal{B}_k^{S'}, \mathcal{B}_k^{S}) \geq d'_{\mathcal{B}}(\mathcal{B}_k^{S'}, \mathcal{B}_k^{S})$.

If the index was constructed with $d'_{\mathcal{B}}$ but searched with $d_{\mathcal{B}}$, then the distance between a pivot and the query may indicate that the pivot's partition cannot include any match, which may be a wrong conclusion since the pivot's covering radius was computed with $d'_{\mathcal{B}}$.

This issue is visualized in Fig. 39, where the distance between the pivot p and the unlabeled, black data points represent the distances computed with the metric $d'_{\mathcal{B}}$, and the according covering radius $r'(p)$. Searching with $d_{\mathcal{B}}(p, q)$ in the figure, yields that the partition of p is not relevant, as the spheres around query and pivot do not intersect, i. e., $d_{\mathcal{B}}(p, q) > r(q) + r(p)$. However, if we computed the probabilistic distances from the pivot to indexed elements, represented by the gray data points and the covering radius $r'(p)$, the search spheres would intersect, i. e., the partition of p contains matches. Hence, to correctly determine whether a partition can be excluded, the auxiliary distance needs to be employed.

Combination of distances

In order to incorporate the alignment confidence despite this issue, search must use a combination of the metric $d'_{\mathcal{B}}$ and the probabilistic distance $d_{\mathcal{B}}$. For *range queries*, the metric $d'_{\mathcal{B}}$ is used to obtain

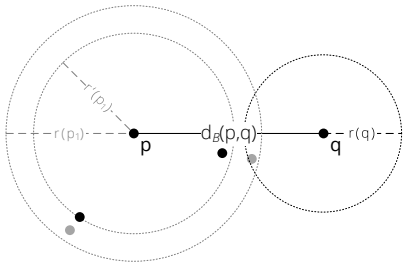


Figure 39: Search in metric spaces: Probabilistic distance may exclude relevant partitions.

potential matches from the index structure and distance $d_{\mathcal{B}}$ to verify whether they are matches and to rank them. Since $d'_{\mathcal{B}}$ underestimates the probabilistic distance of relation sets, searching with $d'_{\mathcal{B}}$ will not exclude partitions that contain process models whose probabilistic distance satisfies the search radius. Conversely, the search result obtained with $d'_{\mathcal{B}}$ may contain false positive matches that are beyond the search radius in case of range search. This is accounted for by verification of the search result, i. e., excluding matches for which holds $d_{\mathcal{B}}(\mathcal{B}_k^{S'}, \mathcal{B}_k^{S}) > r(\mathcal{B}_k^{S'})$, where $\mathcal{B}_k^{S'}$ is the query relation set and $r(\mathcal{B}_k^{S'})$ the search radius.

For search with a *nearest-neighbor query* that requests n matches, the radius is not static and reduced stepwise. This makes verification of the result set as introduced above unfeasible. If some matches were excluded by verification as post-processing the result set, it could contain less than n elements. Therefore, the probabilistic distance must be used during search, to manage the result queue. If, during search, a relation set $\mathcal{B}_k^{S}$ is found whose metric distance $d'_{\mathcal{B}}(\mathcal{B}_k^{S'}, \mathcal{B}_k^{S})$ satisfies the search radius $r(\mathcal{B}_k^{S'})$, its probabilistic distance $d_{\mathcal{B}}(\mathcal{B}_k^{S'}, \mathcal{B}_k^{S})$ will be computed and if this still satisfies the search radius, it will be inserted into the result queue at a position according to its probabilistic distance to the query. The search radius is computed by means of the auxiliary distance from the query to the most distant element in the result queue $\mathcal{B}_k^{S^n}$, i. e., $r(\mathcal{B}_k^{S'}) = d'_{\mathcal{B}}(\mathcal{B}_k^{S'}, \mathcal{B}_k^{S^n})$. The comparison of pivots with the query also uses $d'_{\mathcal{B}}$.

This procedure ensures that the result queue contains only the n most similar relation sets, according to $d_{\mathcal{B}}$, whereas no potential matches will be excluded during search.

Part III

EVALUATION AND DISCUSSION

7 EVALUATION

This chapter is based on results published in [153, 158, 157, 110, 155, 160].

Research in the field of business process model search has created an abundance of promising techniques toward discovering commonalities and equivalences. To configure such techniques for optimal performance and transfer them into practice, they need to be evaluated with respect to their quality and performance. This offers the opportunity to compare different techniques with each other, and identify strengths and weaknesses.

Following our research objective, we provide a comprehensive evaluation of the quality, i. e., the capability to obtain and present relevant search results, and the performance, i. e., resource consumption and scalability, of our techniques. Therefore, we first introduce a methodology to evaluate process model search in Sect. 7.1 and present the material and background of our evaluations in Sect. 7.2. Based thereon, we evaluate the presented techniques for querying in Sect. 7.3 and for similarity search in Sect. 7.4. In Sect. 7.5 we assess the effectiveness of the quality measures for search results that have been presented as part of our framework.

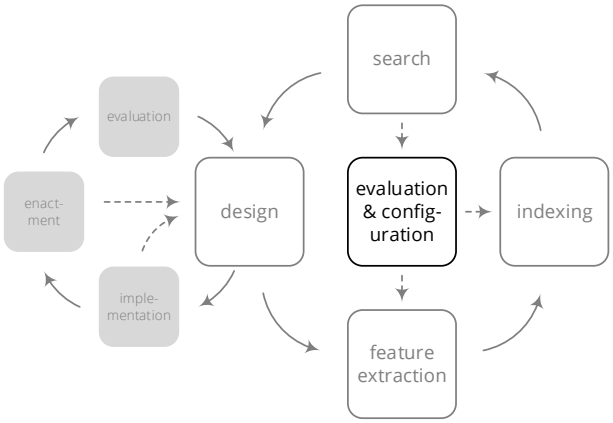


Figure 40: Topics addressed in this chapter: Evaluation and configuration.

7.1 METHODS FOR EVALUATING PROCESS MODEL SEARCH

In Sect. 5.2 and Sect. 6.2, we have elaborated on a large number of approaches to process models search. In order to make these approaches comparable and transfer them into practice, they need to be evaluated with respect to their quality and performance. However, only few authors evaluated their techniques, and different approaches are not comparable [155]. We observed that the reason for this is a lack of commonly agreed evaluation measures and methods for process model search.

In this section, we present evaluation measures borrowed from information retrieval [86, 39, 16] and show how they can be applied to similarity search and querying of business process models, likewise. Therefore, we first focus on quality, i. e., evaluating how meaningful a search result is with regard to the relevance and ranking of provided matches. Second, we turn our attention to the evaluation of the performance of a search technique, i. e., resource consumption and scalability with regard to large process model collections.

We explain the application of evaluation measures by means of diagrams and their interpretation. The examples provided in this section are purely for illustration purposes and do not report on an actual evaluation of our search techniques. The presented measures and methods are applied in the following sections to evaluate both quality and performance of our approach toward process model search by example.

Evaluating the quality of a process model search technique requires a reference to be compared with, also referred to as "standard" or "gold standard" in information retrieval [86, 16]. Such a reference defines, which candidate models ideally match a given query. Quality measures evaluate the difference between an actual search result and the provided reference and, hence, assess how well the search technique complies with the reference. These measures allow for a direct comparison of the strengths and weaknesses of various process model search techniques with regard to a given reference.

We distinguish quality measures by three categories: *Effectiveness* addresses evaluation measures adopted from information retrieval, whereas *robustness* and *ranking* are introduced for searching process models.

Effectiveness

Effectiveness judges on the accuracy of a search technique by the correlation of its output with the reference. Precision and recall, introduced in [304], are probably the most prominent effectiveness measures. Figure 41 illustrates the underlying concepts. Given a process model collection C and a process model query, a particular search technique will propose A as a search result. In contrast, R represents

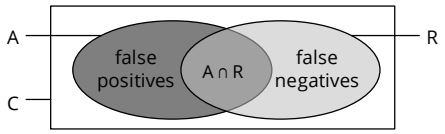


Figure 41: Effectiveness of search result.

the ideal search result defined by the reference, which we refer to as relevant search results, hereafter. Effectiveness measures quantify the agreement between these two sets. $A \cap R$ denotes the subset of matches provided correctly by the search technique with respect to the reference R, and therefore, $A \setminus R$ are considered false positives: matches that are not confirmed by R.

Precision & recall

Precision, p, assesses the amount of false positives in the search result A. The less false positives are contained, the higher is the precision of a search result.

$$p = \frac{|A \cap R|}{|A|}$$

Recall, r, evaluates the amount of completeness of a search result, i. e., the ratio of how many relevant search results are contained in A.

$$r = \frac{|A \cap R|}{|R|}$$

Variations of precision and recall that focus only on a subset of a search result are n-precision, $p(n)$, and n-recall, $r(n)$. These measures require the result set to be ranked. $p(n)$ and $r(n)$ then compute the precision and recall of the first n models of the result set, respectively.

Precision and recall are of limited use to compare the effectiveness of a set of search techniques with each other, as one may have a higher recall and a lower precision than the other, such that no clear preference can be obtained. Therefore, evaluation measures that combine precision and recall provide better comparability.

The *f-score*, f, computes the harmonic mean between precision and recall. It is typically used to evaluate different configurations of a system, where the highest produced f-score indicates the best configuration.

$$f = 2 \cdot \frac{p \cdot r}{p + r}$$

The *overall* measure, o, evaluates the effort to correct a search result so that it resembles the reference. If more relevant models must be added than are already contained, it will provide a negative value.

$$o = r \cdot \left(2 - \frac{1}{p}\right)$$

These measures can only be computed for a result set A of fixed size. This is straightforward in the case of querying process models, as the result set is strictly constrained by the query. However, in case of similarity search, the result set is typically only limited by the number of the models presented to the user or a minimal similarity

threshold. Hence, the size of the result set needs to be chosen a priori. Then, *f-score* and *overall* can be measured for the result set.

On the other hand, similarity search offers the opportunity to expand the search result by adding process models with lower similarity, or greater distance, to the query. While this will include more false positives in the result set, it will also add true positives, and hence, increase the recall of a search result. A single-figure indicator for the effectiveness of a similarity search technique, in the same fashion as the f-score or overall for querying, is the *average precision*, p_{avg}, which denotes the average precision of a search result, after all relevant models have been included in the result set [39].

$$p_{avg} = \frac{\sum_{i=1}^{|C|} (p(i) \cdot rel(i))}{|R|}$$

In the above formula, function $rel : \mathbb{N} \mapsto \{0, 1\}$ determines whether the i^{th} match is relevant, i. e., $\Omega(i) \in R$ and returns 1, otherwise 0.

Ranked search results in general, and similarity search results in particular, lend themselves to an appealing way of expressing a combined figure for precision and recall. *Precision-recall curve* diagrams depict the precision of a search result over increasing recall levels. As more models are added that have a lower ranking, the precision of the search result typically declines. In an ideal case, precision remains high over large recall levels, whereas inferior cases are identified by an early and gradual decline. Figure 42 shows an example for such a diagram, where the precision declines already for comparably low recall values. The slight increase in the curve after a recall level of 0.4 is due to the computation of average values over a set of queries.

Aggregation of results

Above effectiveness measures are computed for a single query and search result only. A thorough evaluation, however, shall consider a diversity of independent queries addressing various aspects of matching. Hence, above measures are often aggregated over a set of queries. For example, aggregation of the average precision over all queries[1]

1 Information retrieval refers to this method of aggregating effectiveness measures as *macro average* [188]

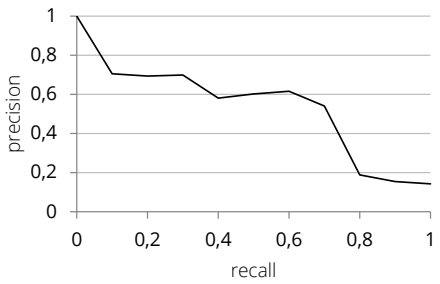


Figure 42: Example precision-recall curve.

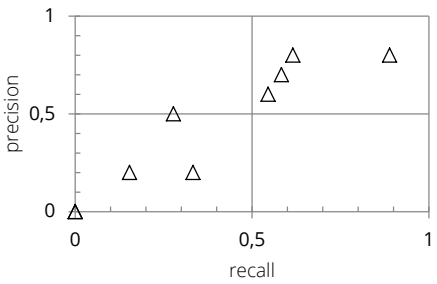


Figure 43: Example precision-recall cluster.

yields the *mean average precision*, which is useful to assess the effectiveness of a similarity search technique in a single figure.

Aggregation, however, hides interesting outliers that should be rather highlighted and critically discussed as specific aspects of a search technique. Plotting the individual performance of queries in a coordinate space spanned by precision and recall offers insights into outliers and provides a basis to discuss their relative impact on the effectiveness. An example is illustrated in Fig. 43. Data points in the lower left quadrant should be discussed for apparent issues, as they show low recall and precision values, i. e., $p < 0.5$, $r < 0.5$.

Robustness & Configuration

One particular aspect of process model search is the construction of a process alignment, before a match is decided, cf. Sect. 3.4. We observed that the quality of an alignment has a major impact on the quality of a search technique as a whole. However, some techniques are more robust against noise in the alignment than others. To this end, *robustness* expresses the insensitivity of a search technique to noise, i. e., reliable results are provided even for inferior alignments. Therefore, an effectiveness measure, e. g., the f-score for querying or average precision for similarity search, is computed over various configurations of an alignment.

Robustness

In the example, shown in Fig. 44, a threshold-based matcher has been used to compute the alignment, i. e., only if the matching fitness of a pair of actions is higher than the threshold, they are considered for an alignment. The curves shows that the mean average precision for various alignment threshold values between 0.4 and 1 rises for increasing threshold values as falsely identified correspondences are eliminated.

Whereas effectiveness measures evaluate the overall quality of a search technique with particular data sets, robustness discloses valuable insights into the very aspect of process matching. Similar evaluations of robustness are possible over all configuration parameters and can be used to configure a search technique along a set of parameters to provide optimal quality.

Configuration

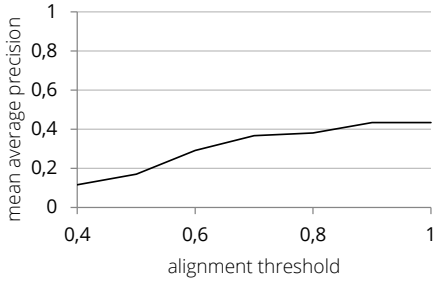


Figure 44: Example robustness.

Ranking

Above effectiveness measures evaluate the accuracy of process model search techniques, but neglect the ordering of matches in the search result. Yet, more relevant process models should be ranked higher than less relevant ones. In the literature, we observed that an evaluation of the search result ranking is rare, as attention is rather spent on high effectiveness of search results in related work.

A set of various measures has been proposed in information retrieval, e. g., the Spearman coefficient and Kendall's Tau, cf. [16], and the ranking agreement measure [110] that allows comparing more than two rankings at once. Here, we resort to a comparably simple, yet expressive measure, the *overlap*.

Overlap

$$l(n) = \frac{|A_n \cap R_n|}{n}$$

In the formula, A_n represents the subset of A that contains the n highest ranked process models; the same holds for R_n. The overlap assesses the agreement of the reference with the actual search result on the n most relevant process models in the search. The overlap is useful to illustrate the quality of a ranking over an increasing size of the data set.

This is visualized in Fig. 45. The measure starts comparably high and then declines, which indicates that the search technique agrees with the reference for the very first matches, but then deviates. As more and more matches are added to the search result, i. e., n rises, the overlap increases and converges with 1, as at some point all matches from the reference will be contained in the search result. In practice, n should be chosen such that it represents the number of process models that are presented to a user on the first result page [16], e. g., between 5 and 10.

Efficiency

The *performance* of a search technique can be estimated by means of a benchmark, i. e., a collection of candidates and queries that are

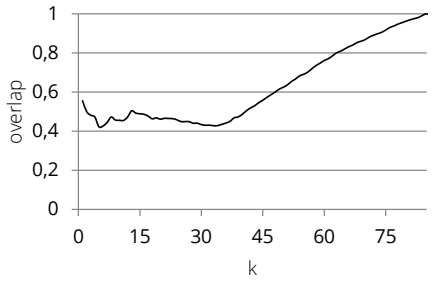


Figure 45: Example overlap.

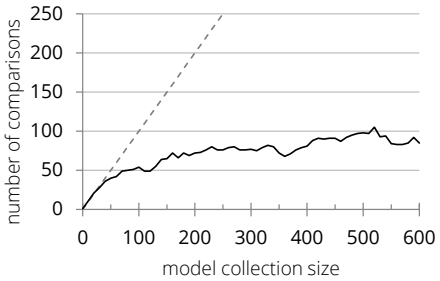


Figure 46: Example efficiency.

searched and for which performance figures are recorded. In literature, we observed absolute performance measures with regard to search time, e. g., [105, 134, 75], and memory consumption, e. g., [133, 136]. These results are of limited use, as they have been obtained in a particular environment and are therefore hardly comparable. Nevertheless, absolute values are required to estimate the general applicability of a search technique.

Efficiency refers to relative measures with respect to resource consumption or computational complexity that are independent of the actual processing environment. These measures address the optimization of the overall search performance that are achieved by optimizations, estimations and heuristics, and additional data structures, e. g., indexes. Here, we focus on the latter. Indexes are employed to exclude candidates from exhaustive comparison, cf. Sect. 2.4, and their benefit can be measures simply by comparing them against sequential search. *Saved comparison operations*, c, is a relative measure that evaluates the performance improvement gained from an index, i. e., the actual number of comparisons, $|C|$, required for search in relation to sequential search, where a query is compared against each candidate, with n being the number of candidates that were not excluded.

$$c = 1 - \frac{|C|}{n}$$

This measure can be aggregated over a set of queries to be more expressive, and a quantitative evaluation of the distribution, e. g., using box-plots, yields valuable insights. For instance, a mean value that is significantly larger than the median over the set of candidates and queries indicates that there exist considerably poor cases that show low efficiency of the indexing approach. Such cases should be examined. Also, variance and standard deviation provide a well-established means to judge on the distribution of values.

Scalability

The *scalability* of a particular search technique can be evaluated by comparing performance figures, both absolute and relative, over an increasing size of the process model collection considered for search. The progression of the resulting curve gives an insight into the capacity of the approach to cope with large process model collections.

An example is illustrated in Fig. 46 that visualizes the performance of a hierarchic index [52] (solid line) compared to sequential search. The figure shows that the number of comparisons is significantly reduced by the index. The index also scales well, as its efficiency increases with growing sets of process models, indicated by the logarithmic progression of the curve.

7.2 MATERIAL

To evaluate the search techniques presented in Chap. 5 and Chap. 6, we conducted a series of experiments using reference data to judge on the quality and the performance of searching process models by example. Before proceeding to the results of these evaluations, we briefly elaborate on the data used for our experiments.

Reference Process Model Collection

We leveraged the *SAP reference model* [60], a collection of EPC process models from industry. Published by business software vendor SAP in 1997, it comprises 604 EPC process models which represent reference processes implemented in the SAP R/3 system. The processes of the SAP reference model capture different functional aspects of an enterprise, e. g., sales and accounting. A comprehensive discussion of the structural and behavioral characteristics of the SAP reference model can be found in [203]. These models have been used, for instance, for formal error analysis [299, 203], analysis of business process modeling [158], extraction of reusable action patterns [264], merging of process models [204, 249], and label analysis [166]. Approaches toward efficient search in large process model repositories [133, 153, 328] rely on this collection, as few other available collections contain as many models. In short, these models became the reference for empirical research on model collections in the last ten years.

Benefits of the reference model

The choice to use models from this collection is motivated by two aspects in particular. On the one hand, a variety of models in this collection show a functional overlap and, thus, it qualifies for experiments on querying process models. Yet, the models use a homogenous vocabulary, i. e., identical concepts are referred to by the same terms, and show only few differences in their spelling, which makes the construction of an alignment straightforward. Consequently, this collection allows us to focus on the evaluation of the actual search technique limiting the bias that is introduced by an alignment of two process models. At the same time, the SAP reference model describes business processes that are representative in size and complexity. Models consist of 21 nodes on average, with up to 130 nodes per model. Less than eight percent of the models comprise 50 nodes or more.

On the other hand, these models have repeatedly been used for empirical research, e. g., to evaluate the effectiveness of process model similarity measures with regard to human assessment [75, 158]. Thus, for the evaluation of similarity search, we could resort to this additional data set, already.

Issues of the reference model

Since we focus on the behavior of process models in terms of successor relations to compare candidate and query process models during search, we had to exclude models that showed syntactic errors or ambiguous instantiation semantics, as both issues prohibit deriving correct successor relations.

The latter issue refers to a specific problem of the execution semantics of EPCs [67]. In EPCs instantiation of a process can be modeled by a combination of start events, i. e., events that have no incoming sequence flow. Instantiation of processes with multiple start events is decided by the semantics of how these events are combined, e. g., if a pair of start events is joined with an AND connector, they need to occur both to instantiate the process. However, arbitrary combination of different connectors may lead to situations in which process instantiation cannot be decided unambiguously anymore, i. e., in case of a missing start join [67]. A start join is a connector such that from every node in the net there is a path to the join connector or one that originates from the join connecter. Consequently, there exists exactly one node in the net that allows determining, whether required instantiation conditions are met.

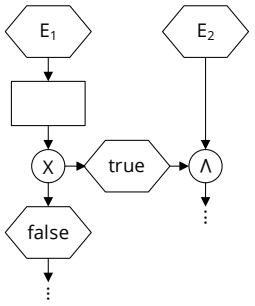


Figure 47: Missing start join.

Figure 47 shows an example of a missing start join from [67]. The semantics imposed by the connectors allows starting a process, if only event E_1 has been observed, or requires that events E_1 and E_2 need to be observed both if the condition of the XOR connector is true. Obviously, this cannot be decided from observing E_1 and hence the instantiation semantics of this process are not clear.

We argue that this is an issue for the practical applicability of the process and should be resolved in the process model. This issue is specific to EPC, as other process modeling languages are more specific in their instantiation semantics. In BPMN, for instance, start events are mutually exclusive, unless they are explicitly connected by an instantiating parallel gateway, which then represents the start join [225].

Construction of a Process Alignment

Finding correspondences and constructing an alignment of a pair of process models is an instance of the assignment problem: From both models, pairs of actions need to be found such that an action of one model matches with its best fitting partner of the second process model. To measure the fitness, or confidence, of a correspondence, we compute the similarity of the labels of these two actions. As we reported above, the SAP reference model collection qualifies for this in particular, because actions show a consistent reuse of action labels across models, which reduces the bias of heterogeneous labels for discovering correspondences.

To find correspondences, we therefore leveraged the normalized string edit similarity, cf. Not. 6.1, as a basis to compute the similarity of action labels. As a preprocessing step, we removed special characters, normalized diacritics and whitespace characters, and tokenized actions into terms by splitting them into consecutive words. Then, we computed the optimal alignment of these terms by the normalized string edit similarity. Then, the confidence of a correspondence is computed by the average similarity of all tokens.

Tokenization enables finding a match between action labels with inverse word order, such as "invoice release" and "release invoice", which would otherwise have an approximate similarity of only 0.2 despite their high similarity. For the process model collection, this strategy proved effective. In order to guarantee a certain quality of correspondences, we require a minimum similarity threshold of these labels, denoted by θ. Only if the similarity of a pair of action labels exceeds this threshold, it will be considered for a correspondence.

7.3 QUERYING PROCESS MODELS BY EXAMPLE

Despite the abundance of approaches that present expressive query languages, cf. Sect. 5.2, process model querying has not yet been widely adopted in practice. In particular, due to the lack of tool support, there are no studies toward the usefulness and relevance of provided solutions. In this section, we analyze the applicability of *querying process models by example*, cf. Chap. 5, in an experiment with regard to its quality and performance.

Experimental Setup

Reference data

As a consequence of the lack of experimental evaluation of process model querying approaches, no reference data has been collected that could be used. Therefore, we set up an experiment to measure how well abstract trace inclusion and closeness correlate with human assessment. For this experiment, we chose 34 candidate models of the SAP reference model [60]. From the candidate models, we manually

generated ten query models and paired each query model with ten of the candidates. Seven process modeling experts were then asked to determine for each pair of query and candidate process model, whether the process model matches the query and to rank all matches on a Likert scale from 0 to 6, where 0 indicates no match and higher values are correlated with increasing relevance. To decide a match, the experts were given the guideline that "a matching process model shall be able to replay the behavior of a query". For each query, we computed the list of matching models that showed abstract trace inclusion and ranked them by their closeness.

In a second experiment, we assessed the performance of our approach and its scalability as follows. For increasing sizes of a base model collection, we searched for 100 process models chosen as queries and measured the median search time over these search runs. Candidates and queries were chosen randomly from the set of sound free-choice process models of the SAP reference model collection [60] to account for various levels of complexity when processing queries. We conducted each search run twice with the same queries respectively: first as a sequential search, where we compared the query with each model in the current collection, and second as efficient search by means of the filter and verification approach using the inverted index, presented in Sect. 5.5.

We implemented query matching, cf. Sect. 5.3 and closeness computation, cf. Sect. 5.4, in Java using the jBPT library[2]. For the inverted index, cf. Sect. 5.5, we adopted Apache Lucene[3]. We extracted all eventual successor relation pairs of a process model, i. e., $>^{S}_{b(S)}$, and used these as index terms, referencing the process' net system and its minimal-k-successor relation. In the filtering phase, we extracted all 1-successor relation pairs from a query, i. e., $>^{S'}_{1}$, and searched for a conjunction of these terms.

All experiments have been carried out on a 2.8 GHz CPU running a Java 7 virtual machine with 1 GB of heap memory assigned, of which only 400 MB were used. For sound, free-choice workflow systems, minimal-k-successor relations have been precomputed and cached in memory to exclude times for successor computation and I/O operation during search. This is in line with the presented indexing approach, where these relations can already be computed and stored when the model is added to the index.

Configuration & Robustness

Our approach to querying process models by example employs a rather strict approach to decide a match and, therefore, offers no configuration parameters to influence deciding a match or computing closeness. However, at its basis, it employs an alignment to identify

2 http://code.google.com/p/jbpt/

3 http://lucene.apache.org/core/

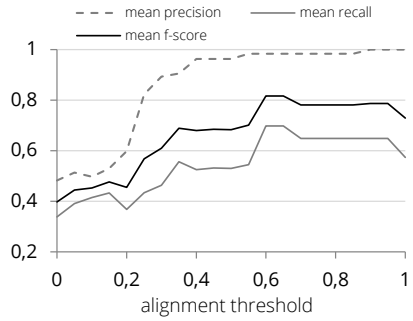


Figure 48: Precision, recall, and f-score for varying alignment thresholds.

corresponding actions in the query and candidate models. In order to obtain an optimal alignment, the threshold θ has been introduced that determines the minimum similarity of labels to consider a pair of actions for the alignment.

Figure 48 shows the average values for precision, recall, and the f-score over all ten queries for increasing alignment thresholds. These values generally increase from 0 to 0.6 as a low threshold leads to inappropriate correspondences, i. e., actions that are not related are identified to represent the same unit of work. Consequently, traces of the query that would be included in the traces of a candidate will not be discovered. At threshold values of $0.6 \leq \theta \leq 0.65$, the mean f-score reaches its local maximum of 0.817 and thereby identifies the best trade-off between precision and recall. Beyond this value, recall deteriorates, as small differences in the labels are not being compensated for and fewer models are discovered to be a match. At the same time, precision rises, which points out that our approach produced correct results.

Only, when the threshold value is raised to 0.85, the precision becomes 1, i. e., no models that are excluded as a match by humans are suggested by the query technique. The reason for this is found in one particular query-candidate-pair that is provided as a false match for $\theta = 0.65$, due to the labels *difference processing* in the query and *preference processing* in the candidate. While these are different tasks, confirmed also by human assessment, their string similarity, cf. Sect. 7.2, is $0.84\overline{9}$ and they are falsely identified as a correspondence for lower thresholds. However, a similarity threshold above 0.85 should not be chosen, as this comes at the cost of a loss in recall. To evaluate effectiveness and ranking of our approach based on the experimental data, we therefore choose the alignment threshold $\theta = 0.65$.

Effectiveness

With the above threshold, the mean average precision of querying process models by example compared to the obtained human assessment is 0.679 with a mean precision of 0.983 and a mean recall of 0.698. This recall value is acceptable, considering that querying decides matches

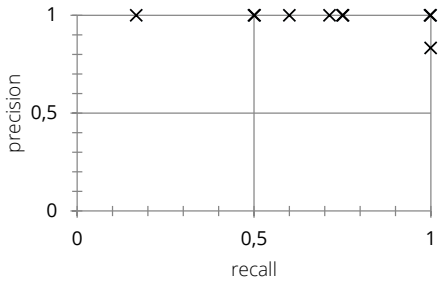


Figure 49: Precision-recall diagram for ten queries. Each data point represents one query, data points with identical recall and precision values appear bold.

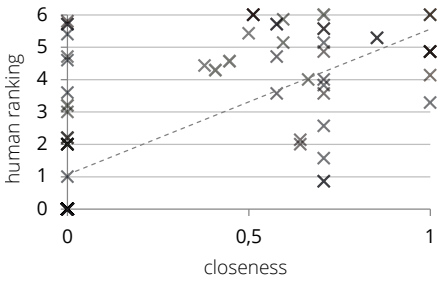


Figure 50: Obtained closeness values relative to the average human ranking. The dashed line indicates linear regression over all ten queries.

by precise semantics, whereas we experienced that humans are more forgiving when comparing process models. This is supported by the respective precision and recall values for each query, depicted in Fig. 49. Deciding a match by abstract trace inclusion yields consistently high precision values, i. e., safely excludes false positives. The single data point that shows a lower precision (0.833) is due to the similar labels mentioned above. On the other hand, the recall values indicate that abstract trace inclusion is more selective than humans as it is able to decide the inclusion of behavior precisely. However, for all but one query, at least the majority of matches proposed by the experts are confirmed by abstract trace inclusions.

We found the cause for this negative outlier in one particular query that obtained a recall value of only 0.167. The concerning query features two actions that emerge from a parallel gateway, which requires a match to execute these actions in parallel or interleaved ordering as well, cf. Sect. 5.3. However, several subjects considered process models as a valid match, if they allow executing both actions in at least one of the possible orders. Hence, for some types of queries, also partial match results appear to be relevant, an aspect that shall be addressed in future work.

Ranking

As our experiment has been limited in scale, the overlap does not provide useful insights into the correlation between closeness and human based ranking. We therefore resort to another visualization of the result, depicted in Fig. 50. Here, we projected each match for a query in a coordinate diagram spanned by the human based ranking and closeness. With respect to the closeness based ranking, the diagram shows that high closeness values are obtained for models that are also ranked high according to human judgement. Figure 50 illustrates the least-square linear regression over all queries as a dashed

line. This underpins the trend of high closeness values being obtained for models that are also ranked high by humans. For our comparably small data set, however, these results turned out to be not statistically significant.

To understand in which cases the closeness measure approximates the human ranking well, we conducted a detailed inspection of the models and their rankings. We observed that models requiring less effort to complement the query were often ranked higher than more complex models. As such, the appropriateness of the closeness measure for ranking models that show significant differences in their sizes has to be further investigated in future work.

Efficiency

Figure 51 shows the required search time in milliseconds for sequential search (dashed gray curve) and efficient search (solid curve) on a logarithmic scale. To account for the similarity of strings we accepted keys in the inverted index with a similarity higher than θ compared to the successor relation pairs of the query.

Sequential search

The curve for *sequential search* is provided for reference. The strong raise until 50 models results from the choice of query models. In these cases, less than half of all query models are contained in the collection. Hence, in many cases no or only few candidates are found that share correspondences with the query and qualify for deciding a match. For larger data sets, Fig. 51 indicates linear growth from 12 to 30 seconds median search time over the increasing process model collection. At a data set size of 450 models, sequential search consumed approximately 15.5 seconds on average. The negative peaks in the median sequential search time emerge from the complexity of the chosen queries. Some queries were trivial or rather simple. If many such queries were chosen for search at a certain collection size, the median search time dropped significantly.

Efficient search

The inverted index shows a significant gain in search speed whose median remains below 25 milliseconds for all 450 models in the collection, the maximal average search time we encountered was 551

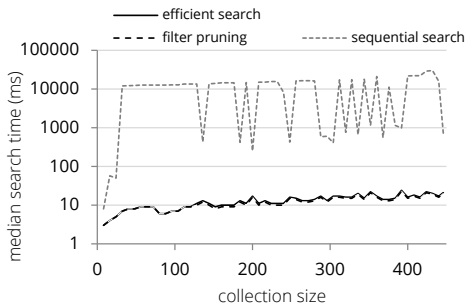


Figure 51: Search efficiency over increasing size of a model collection for sequential (dashed curve) and efficient (solid curve) search.

milliseconds. With these numbers, the index shows a performance of two orders of magnitudes faster than sequential search. Filter pruning (black dashed curve in Fig. 51) shows the amount of time that has been used by the inverted index to exclude models in the filter phase. The difference between the dotted and the solid curve represents the amount of time required for verification. The inverted index proves very effective in excluding invalid candidates early as the median verification time constitutes only few milliseconds, expressed by the affinity of the curves for efficient search and filter pruning.

7.4 SEARCHING PROCESS MODELS BY SIMILARITY

In Chap. 6, we introduced process model *similarity measures* based on successor relations of their actions, as a means to search for process models by a notion of behavioral similarity that evaluates the ratio of shared relations of the models' relation sets. In this section, we examine the proposed measures with regard to their quality and performance.

Experimental Setup

The SAP reference model was used in [75, 158] to investigate the quality of various distance measures for process model search based on a reference data set that comprises human assessment of the similarity of models. The reference data set, presented in [75], consists of 100 randomly selected models from the SAP reference model, referred to as candidate models. Out of this set, 10 models have been selected to be used as query models. These queries partly underwent manual changes. Two models were left unchanged, whereas eight models have been slightly changed. Examples of these changes are the extraction of a subgraph or an adaptation of the behavioral type of control flow routing elements.

Reference data

For pairs of candidate query models, the data from [75] provides a relevance mapping. The authors and 20 process experts assessed the similarity of every pair of query and candidate model on a Likert scale from 1 to 7. Pairs with a score of 5 or higher indicate that a candidate model is relevant for the query [75]. As such, the data set allows quantifying the effectiveness of similarity measures. Note that from the 100 models of this test set, 15 had to be excluded for our experiment because of ambiguous instantiation semantics, cf. Sect. 7.2. Since one of these models was a query model, the setup for investigating the quality of process model similarity search includes 85 candidate models and nine query models in our experiment. In our evaluation, we rely on the test set for the "evaluation with homogeneous labels" [75], where query models reused the vocabulary of the candidates. We opted for this test set, because the construction of an

alignment is not in the focus of our work and it allowed us to resort to a syntactic label similarity, described in Sect. 7.2.

We implemented efficient similarity search, cf. Sect. 6.5, in Java using the aforementioned jBPT library[4] for the computation of successor relations and relation sets, and the similarity of these relations. For indexing we resorted to the XXL library[5] [27]. The experiments were carried out on a 2.4 GHz CPU, running a Java 6 virtual machine with 1.5 GB of heap memory assigned. However, only up to 150 MB were required to store the complete index. The index has been kept in main memory to exclude bias for I/O operations during search.

Configuration & Robustness

In Sect. 6.3, we introduced the probabilistic up-to-k-successor relation similarity, sim_Φ, cf. Def. 6.4, that is based on comparing pure successor relations of process models. This has been extended in Sect. 6.4, by the explicit distinction of the behavioral relationship of pairs of actions, i. e., exclusiveness, strict order, and interleaving order, to define the k-relation set similarity, $sim_\mathcal{B}$, cf. Def. 6.7.

Both similarity measures are subject to two configuration parameters, (a) the look-ahead k that determines the length of firing sequences from which successor pairs are obtained and (b) the alignment $(\backsim, \phi)$ used to compare process models. Regarding the latter, we have discussed the matching techniques applied for our experiments in Sect. 7.2. However, the alignment threshold θ that determines the minimum similarity of labels to consider a pair of actions for the alignment needs to be determined with regard to the provided data set.

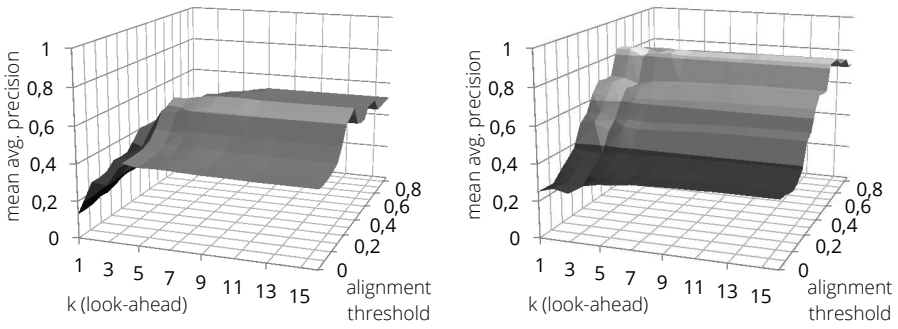
Therefore, we computed search results for the nine queries over increasing values for k and θ. For each combination of the parameters, we computed the mean average precision of the nine search results as an aggregate measure for the quality of a similarity measure. None of the candidate or query models had more than 21 transitions, which leads to a computational successor bound of 441. However, for $k \geq 7$ the successor relations of the majority of net systems remained the same. Consequently, the mean average precision of search converges for these values. This is illustrated in Fig. 52 that shows the progression of the mean average precision over increasing values for the look-ahead k and the alignment threshold θ. The horizontal axes depict the values for the parameters, $1 \leq k \leq 15$ and $0 \leq \theta \leq 1$, and the vertical axis the mean average precision achieved over the nine queries. Here, $sim_\mathcal{B}$, depicted in Fig. 52a, has been configured with equal weights for elementary similarities w_+, $w_\rightarrow$, and $w_{||}$.

Look-ahead

For sim_Φ the mean average precision is significantly low for small k, illustrated in Fig. 52a. This is due to the short look-ahead which

4 http://code.google.com/p/jbpt/

5 http://code.google.com/p/xxl/



(a) Configuration of sim_{Φ}.

(b) Configuration of $sim_{\mathcal{B}}$.

Figure 52: Mean average precision of behavioral distance measures over different configurations for look-ahead parameter k and alignment threshold θ.

yields a match for only few models, i. e., if aligned actions are in close proximity in the traces of both models. With increasing k, more common successors can be discovered, which results in higher precision. In comparison, the mean average precision of $sim_{\mathcal{B}}$, depicted in Fig. 52b, is only slightly affected by the look-ahead parameter. This is due to the explicit incorporation of the exclusiveness relation, i. e., all pairs that are not in successor relations in both models for a given k are also matched, which increases the precision of this measure.

From above observations, we can conclude that larger k increase the precision of the search results, as more actions are considered in behavioral relations. As already discussed in Sect. 4.2, large k may lead to undesired results, if the process models contain cyclic dependencies that span a large fraction of the model.

Alignment threshold

From both diagrams, we observe that the mean average precision improves with an increasing alignment threshold θ up to a certain maximum value. This is expected, as low thresholds lead to falsely identified correspondences. High thresholds, on the other hand, will miss corresponding actions if their labels are not similar enough, e. g., due to typographic errors. In both diagrams of Fig. 52, a local maximum of the mean average precision can be found for $\theta = 0.65$. This thresholds leads to a mean average precision of 0.825 for $sim_{\mathcal{B}}$ with equal weights, and 0.582 for sim_{Φ}. In Fig. 52a, the absolute maximum however, is achieved with an alignment threshold of $\theta = 0.4$, resulting in a mean average precision of 0.655. We attribute this to the overall inferior quality of this measure. Hence, falsely identified correspondences lead to matches that incidentally coincide with the reference data from the human assessment.

Moreover, Fig. 52b shows that $sim_{\mathcal{B}}$ is less subject to biases introduced by low thresholds for the alignment, i. e., for $\theta > 0.35$ the mean average precision is greater than 0.65. This robustness makes it less important to tune the notion of action similarity for a dedi-

	$k = 1$	$k = max$
w_+	0.6643	0.2822
$w_\rightarrow$	0.2208	0.6015
$w_{\|\|}$	0.1149	0.1163

Table 11: Weights for elementary similarity measures of $sim_{\mathcal{B}}$, determined according to Def. 6.8. $k = max$ denotes the successor bounds for each model, respectively.

cated model repository. We conclude that the k-relation set similarity outmatches the probabilistic up-to-k-similarity considerably. Therefore, we will focus on the former for the remaining evaluations, with different configurations for the look-ahead parameter k.

Elementary similarities weights

Computing the weights for the k-relation set similarity according to Def. 6.8 with respect to the 85 candidate and nine query models leads to the results listed in Tab. 11. These numbers show that the majority of behavioral relations for $k = 1$ is of the type exclusiveness, whereas this predominance shifts toward the strict order relation with growing k. The effectiveness achieved with the automatic configuration is discussed below.

In an exhaustive comparison, we also compared various configurations of weights $0 \leq w_+, w_\rightarrow, w_{||} \leq 1$. From the obtained results we observed that for $k = 1$ all combinations with $w_+ > 0$ led to high precision, i. e., neither $sim_\rightarrow$ nor $sim_{||}$ have a significant influence on the similarity quality. This is not surprising, as the majority of transitions are exclusive with regard to the 1-successor relation. For the $k = max$, i. e., using the successor bound for k, optimal results were obtained with configurations that showed $w_\rightarrow > 2 \cdot w_+$. Here, the strict order similarity has the most influence. These observations are in line with the above similarity weights computed from the ratio of relation types, which indicates that the automatic deduction of these weights from a corpus of relation sets proves effective.

Effectiveness

To evaluate the effectiveness of the proposed similarity measures and their derived configurations, we elaborate on the relation between precision and recall. Therefore, we computed the precision of a search result for recall values between 0 and 1 in intervals of 0.05. This is depicted using precision-recall curves in Fig. 53. We compare $sim_{\mathcal{B}}$ with minimal ($k = 1$) and maximal ($k = b(S)$) look-ahead values, respectively, to study both ends of the spectrum that has been indicated in Fig. 52.

Mean average precision

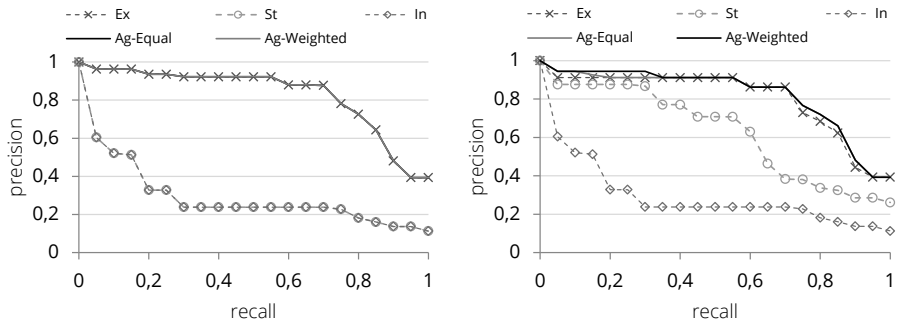
Using the nine query and 85 candidate models, we obtained the following results in terms of mean average precision for $k = 1$.

- Exclusiveness Similarity (Ex): 0.832
- Strict Order Similarity (St): 0.28
- Interleaving Order Similarity (In): 0.28
- Relation Set Similarity with equal weights (Ag-Equal): 0.832
- Relation Set Sim. with computed weights (Ag-Weighted): 0.832

We observe that the exclusiveness similarity yields the best results and shows equal mean average precision with either weighted relation set similarity. Strict order and interleaving order similarity show only inferior results, but do not compromise the overall results due to the low share of these relations in the process models, indicated by the weights in Tab. 11. This observation is confirmed by the precision-recall curve shown in Fig. 53a, where exclusiveness and relation set similarity with either weight are identical. This leads to the conclusion that for $k = 1$ a match is decided merely by the common exclusiveness relations.

For the relations set based on eventual successors, i. e., using the successor bound as look-ahead, we observed that the strict order relation dominates the relations in the process model collections. This is not surprising, as a large look-ahead reveal more ordering relations than a small one, cf. Sect. 4.2. In terms of mean average precision, we observe the following results for similarities with $k = max$, which denotes the successor bound.

- Exclusiveness Similarity (Ex): 0.793
- Strict Order Similarity (St): 0.626
- Interleaving Order Similarity (In): 0.28
- Relation Set Similarity with equal weights (Ag-Equal): 0.821
- Relation Set Sim. with computed weights (Ag-Weighted): 0.828



(a) $sim_{\mathcal{B}}$ with $k = 1$. (b) $sim_{\mathcal{B}}$ with $k = max$.

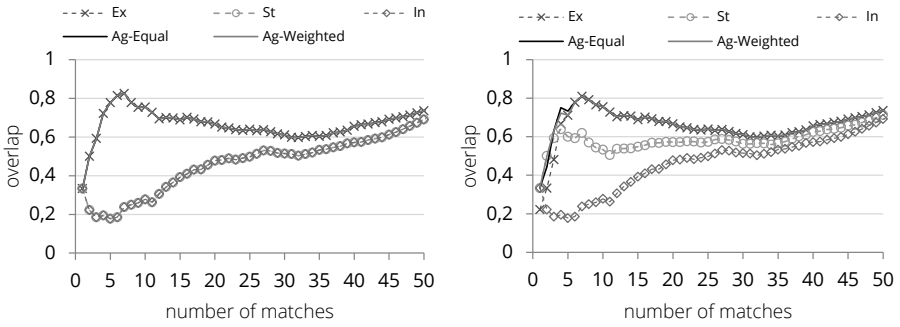
Figure 53: Effectiveness for elementary and aggregated (with equal or determined weights) relation set similarities.

In contrast to the similarities based on $k = 1$, the results for the exclusiveness and strict order similarities are closer in this case. Further, the similarities are complementary, which is revealed by the mean average precision for either aggregated similarity that is higher than for the elementary similarities. The best result, however, is obtained once the similarity is configured with the determined weights, even if the improvement is only minor. Support for this observation is also provided in Fig. 53b, where the relation set similarity with computed weights yields the highest precision over all recall values, compared to the equally weighted and elementary similarities.

Ranking

We evaluate the ranking of the *k*-relation set similarity with regard to the human scoring by means of the overlap that measures the ratio of common matches within the top n matches. Figure 54 shows the results for $sim_{\mathcal{B}}$ with $k = 1$ and $k = max$, where the latter indicates the successor bound obtained for each model, respectively. Both configurations of the *k*-relation set similarity show similar achievements. However, we observe again that, for $k = 1$ the aggregated similarities coincide with the exclusiveness similarity, whereas for $k = max$, the aggregated similarities outperform elementary similarities.

The figures indicate that the ranking of the very first results, i. e., for $n \leq 3$ is low but improves significantly and achieves a high agreement with the human ranking for $4 \leq n \leq 11$ that is above 0.7. For the first three models, humans chose different matches, however, for the top 10 models of the search results, humans and behavioral similarity largely agree. This is in line with an average of nine relevant models per query model provided by the human assessment data set. We observe a decline of the curve due to the higher variety as more models are added to the search result. Eventually, i. e., for $k = 85$ the overlap equals 1, as all candidate models are contained in the search result (not illustrated).



(a) $sim_{\mathcal{B}}$ with $k = 1$.

(b) $sim_{\mathcal{B}}$ with $k = max$.

Figure 54: Evaluation of the ranking of the *k*-relation set similarity by the overlap of the n top matches.

Efficiency

To elaborate on search efficiency, we measured the search performance for nearest-neighbor queries that find the 10 most similar models to the query model, cf. Sect. 6.5. Since this analysis does not require any human input, we were able to use all models with unambiguous instantiation semantics, i. e., 507 models of the SAP reference model collection. The experiments were conducted as follows.

For different sizes n of the data set, i. e., an increasing number of indexed models, we built an M-Tree index [52]. We resorted to the M-Tree, as it has been reported a well-established index structure for similarity search [47, 122]. The M-Tree can be configured by two different parameters: First, the method to *choose a pivot* yields significant differences in the number of comparison operations at search time, as we showed earlier [153]. For our experiment pivots that have a maximum distance to each other (`M_LB_DIST` [52]) are chosen for nodes. While this yields more operations to build the index, it showed significantly better search performance [153]. Second, the node size, i. e., number of pivots per node, is used to configure the M-Tree, cf. Sect. 6.5. As part of our experiments, we examined 2, 5, 10, 20, 50, and 100 pivots per node.

Once the index has been built, we chose a random subset from the indexed models, and searched for the 10 nearest neighbors. For each query posed, we measured the number of comparisons required and the time it took until the search result has been returned. We conducted the experiments for the k-relation set similarity, $sim_{\mathcal{B}}$, whereas we resorted to the auxiliary alignment presented in Def. 6.10 to use a standard implementation for the M-Tree [27].

First, we focus on an evaluation of search efficiency in relative terms, that is, the ratio of saved pairwise similarity computations compared to exhaustively examining the model collection, i. e., comparing the query with every model in the data set. For that purpose, we computed the average of required comparison operations over all queries evaluated against the model collection. Figure 55 shows the maximum efficiency of search runs for various node sizes and similarity measures. The diagram indicates that the efficiency of search

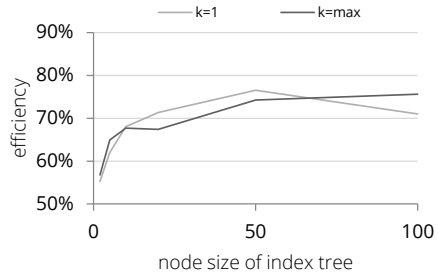


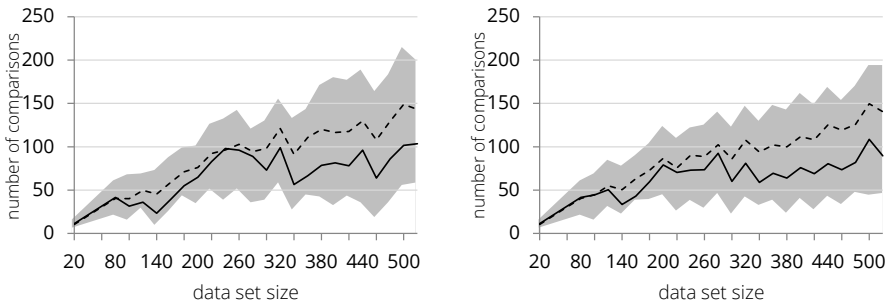
Figure 55: Efficiency of search for k-relation set similarities with $k = 1$ and $k = max$ for increasing node sizes of the index.

increases over the size of index nodes, reaching its maximum value at a node size of 50 in most cases.

Our results indicate a significant gain of efficiency, as search in the M-Tree allows saving at least 75% of comparison operations. The k-relation set similarity with $k = 1$ offers the highest potential for efficiency, cf. Fig. 55. Still, even in the worst cases, i. e., at a node size of 2, both configurations of the relation set similarity save 55% of comparison operations on average. Upon conducting our experiments, we observed rather consistent trends when varying the number of pivots per node in the M-Tree configuration. Thus, we resort to illustrating our experimental data only for the setting of 50 pivots per node, hereafter.

Figure 56 illustrates the search performance in terms of required comparison operations over an increasing size of the model collection that is searched. Depicted is the median (solid line) and average (dashed line) number of comparison operations needed, and, in order to evaluate distribution, the upper and lower quartile, denoted by the boundaries of gray area.

Upon studying the diagrams, it becomes apparent that the average number of comparisons is consistently higher than the median. This indicates a number of high outliers in terms of comparison operations for certain search runs. The distribution of differences, i. e., the gray area between the lower and upper quartile, discloses a high fluctuation of search performance. The reason for both observations is the uneven distance distribution of model pairs in the model collection, with regard to the considered measure. On the one hand, there are few models that are very similar to each other. If more similar matches than n, the desired size of the search result, are present, the search radius cannot be effectively reduced and only few subtrees can be pruned, cf. Sect. 6.5. On the other hand, a considerable fraction of similar models are indeed duplicates. If n is greater than the number of duplicates, many subtrees can be pruned.



(a) $sim_{\mathcal{B}}$ with $k = 1$. (b) $sim_{\mathcal{B}}$ with $k = max$.

Figure 56: Average (dashed line), median (solid line), and lower and upper quartile (gray area) of number of required comparison operations over increasing size of data set.

We discovered the following minimum and maximum savings over the average number of comparisons for $sim_{\mathcal{B}}$. Again, $k = max$ indicates the successor bound.

- Relation Set Similarity, $k = 1$: 47.5%–76.6%
- Relation Set Similarity, $k = max$: 47.5%–74.3%

In general, we observed that the efficiency of the search runs increased with growing size of the data set. Despite the fluctuation in the diagrams, this indicates a logarithmic complexity of the search approach, in terms of comparison operations needed. That is, the index proves more efficient the larger the data set is, because early pruning of partitions has a higher impact. We conclude that the combination of M-Tree indexing and k-relation set similarity enables significant gains in search efficiency in relative terms.

We briefly turn the focus on search efficiency in absolute terms, to see whether the approach yields a practically useful setup. Therefore, we measured the time a search run took to provide a search result. Search times show trends similar to the efficiency in terms of the ratio of saved search operations. The average among all of our experiments yielded a search time of 311ms, whereas more than half of all search results were returned within 100ms. We observed that large query models yielded considerable search times of up to several seconds. This is grounded in the computation of k-relation set similarity. The complete relation set of a model consists of the Cartesian product of their actions, which leads to quadratic computation complexity to the size of the compared models. However, if we consider the accomplished effectiveness and robustness properties of our behavioral metrics, the results are generally within acceptable time ranges.

7.5 QUALITY MEASURES FOR SEARCH RESULTS

In Sect. 4.4, we introduced evaluation measures to assess the quality of a search result in terms of *confidence*, i. e., how much the best search results stand out, *discrimination*, i. e., whether search results can be well distinguished and therefore be ranked unambiguously, and *ranking agreement*, i. e., how much a set of distance measures comply with each other with respect to the ordering of matches.

Experimental Setup

As a basis for this evaluation, we chose three complementary process model distance measures and a human assessment of process model similarity [75] to obtain search results and rankings.

Alignment distance

The *alignment distance* (AD) determines the distance of two process models by the similarity of common actions relative to the number of nodes of a process model. It is rather simplistic and does neither address the graph structure or nor the behavior of a process model.

Definition 7.1 (Alignment distance)**.**
Let $N = (P, T, F)$ and $N' = (P', T', F')$ be two net, $\overline{T} \subseteq T$, $\overline{T'} \subseteq T'$ subsets of their transitions, respectively, and $(\smile, \phi)$ an elementary alignment of their transitions, i. e., $\smile \subseteq \overline{T'} \times \overline{T}$.
The *alignment distance* is defined as

$$d_{\smile}(N', N) = \frac{\sum_{\smile_t \in \smile} \phi(\smile_t)}{|\overline{T}| + |\overline{T'}| - |\smile|}$$ ◂

The alignment distance only considers transitions that have business semantics, cf. Def. 5.2, and abstracts from transitions that have been introduced in the process of transforming business process models to net systems. This is accounted for by $\overline{T}$ and $\overline{T'}$, respectively. The alignment distance yields a similarity notion that belongs to the type feature aggregation, discussed in Sect. 6.2. The alignment $(\smile, \phi)$ is computed according to Sect. 7.2 with a threshold of $\theta = 0.65$.

Structural distance

The *graph edit distance* (GED), introduced in [40], considers a graph's structure and computes the minimal cost to transform one graph into another by atomic graph operations: insert, delete, and substitute nodes and edges. Applied to process models [73], the substitution of nodes coincides with the process alignment; nodes that have no correspondence in one model are subject to deletion or insertion. As the GED computation suffers from unlabeled nodes [40, 213], e. g., gateways, process models have to be reduced to dependency graphs that only consist of actions and edges between them.

The GED implementation in [73] employs a normalized string edit distance [168] for the cost to substitute action nodes and thereby accounts for the confidence of the alignment. To improve similarity, the computation of the GED and the construction of an alignment are carried out in an iterative, integrated fashion, i. e., the alignment is constructed such that it yields a minimal graph edit distance for a pair of models. For our comparison, we therefore relied on similarity data provided with the human assessment of similarity [75].

Behavioral distance

As a behavioral distance measure, we leverage the *k-relation set distance* (RSD), introduced in Def. 6.7, that evaluates the amount of shared behavior in terms of execution order of actions. For its configuration we chose the successor bound as look-ahead parameter k, $\theta = 0.65$ for the alignment threshold, and the automatically determined weights in Tab. 11.

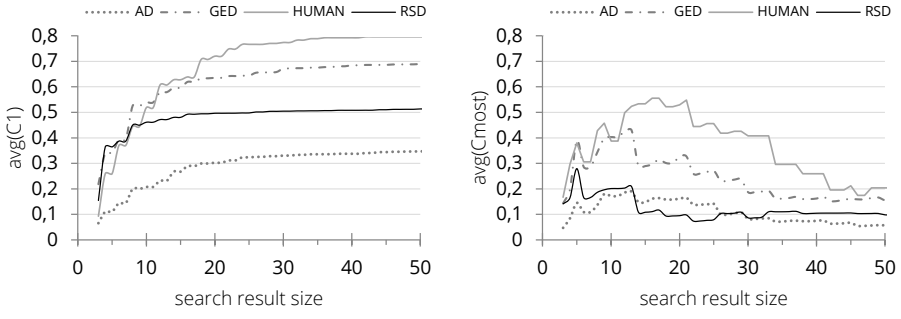
In order to judge on the expressiveness and applicability of our measures, we conducted a set of experiments, for which we use the SAP reference model collection [60], and the human assessment of similarity provided in [75] and introduced in Sect. 7.4. With each of the nine query models and above distance functions, we conducted nearest-neighbor searches with search result sizes of 3 to 50 matches, applied our evaluation measures to each search result and computed the arithmetic mean of these values over the queries. We compared

this with the human judgement of similarity. Therefore, we mapped the average human assessed Likert score for every pair of query and candidate models to distance values $1 - \frac{likert\ score}{8}$ to produce an equal distribution of distance values without the values 0 and 1. We refer to the according data set as HUMAN hereafter.

Confidence

Confidence of a search result evaluates whether the best matches are significantly better than the rest of the search result. Figure 57a shows the confidence of the best match, $\mathcal{C}_1$, cf. Def. 4.6, over an increasing size of search results. For all similarity measures, the confidence increases as more models are obtained. This behavior of the measure is reasonable, because the more inferior matches are retrieved and added to the search result, the worse the reference value $med(\Omega)$, the median distance to the query, gets. Accordingly, the ratio between the minimum and the median is shifting toward the benefit of the best result. However, the measure converges, which supports the intuition that the median is a stable reference value.

Confidence over the most matches, $\mathcal{C}_{most}$, cf. Def. 4.7, also increases for small search results, but reveals a peak and then decreases, as depicted in Fig. 57b. For the algorithmic distance measures, this value is approximately achieved with search result of 10 models, and 15 for the human similarity assessment. This is due to the fact that the measure of interest, $med(\Omega)$, is shifted by the size of the search result. When the peak is passed by adding inferior matches, the majority of search results are not significantly better than all results anymore. From this we conclude that in our model collection the top 5–8 models perform best against the remaining models in the search result, on average. This characteristic can effectively be used to assess the number of meaningful models in response to the query just by the search result.



(a) Confidence best match.

(b) Confidence most matches.

Figure 57: Application of confidence measures to similarity search results, compared with human assessment.

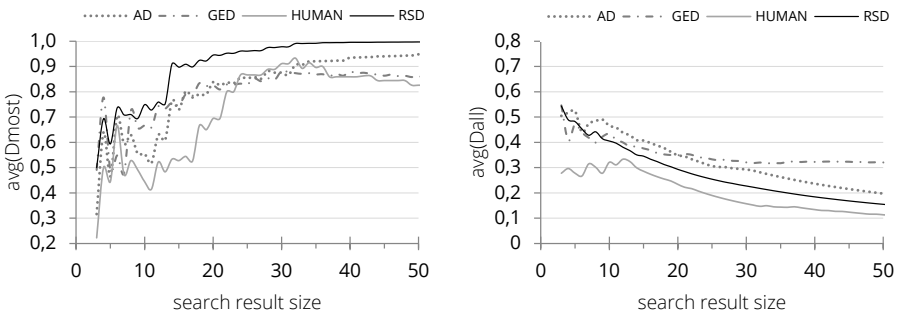
For both confidence measures, we observe that the human assessment (solid grey curve) outperforms algorithmic distance measures, which is not surprising. Nevertheless, the tendency of human assessment agrees with the distance measures in that a high value in confidence indicates a good assessment of relevance of the provided search results.

Discrimination

Discrimination measures evaluate, whether a search result can be ranked unambiguously, where discrimination of most matches, $\mathcal{D}_{most}$, cf. Def. 4.8, in particular considers the superior half of the search result in comparison to all results. Figure 58a shows that this value increases over a growing size of the search result, which indicates that the majority of superior models can be ranked unambiguously. However, a high value of this measure at a large size of results points to the fact that the set of inferior models remains within a rather small distance interval and may not be ranked unambiguously.

This assumption is supported by the discrimination of all results, $\mathcal{D}_{all}$, cf. Def. 4.9. With increasing size of the search result, the majority of results deviate largely from a uniform inter-result distance distribution, that is, the ranking of models after position 20 may not be meaningful anymore and users may need to be supported with additional data to distinguish the models and assess relevance for their search intention.

Diagrams Fig. 58a and Fig. 58b show that the human assessment of discrimination is lower than discrimination provided by algorithmic distances. This can be explained by the fact that the assessment of relevance is subjective to the users and their task. Different users rate the similarity of the same pairs of query and candidate models differently, which leads to a lower agreement among them, and thus a higher ambiguity of the identified ranking. As the human judgement



(a) Discrimination most matches.

(b) Discrimination all matches.

Figure 58: Application of discrimination measures to similarity search results, compared with human assessment.

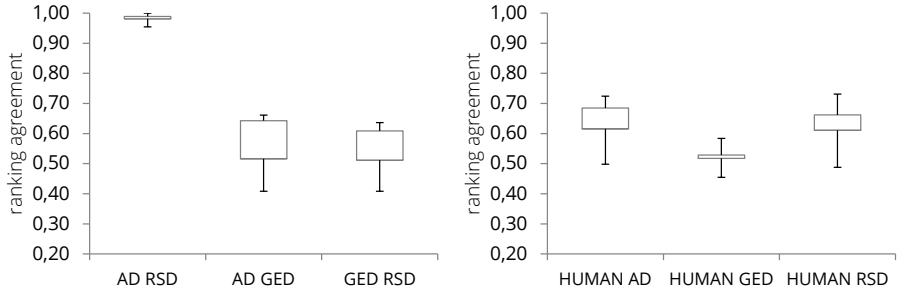
was based on a seven point Likert scale, there were furthermore only seven values to distinguish the similarity of a process model pair.

Ranking Agreement.

The ranking agreement measures, how much a set of distance functions agree on the ordering of a set of process models, e. g., the ordering of a search result. While it can be used to evaluate the ranking of a particular search result with human assessment, it has been devised to compute an agreement value of more than a pair of distance functions and thereby assess the overall ranking of a search result with regard to a number of different distance functions.

Figure 59 illustrates the average ranking agreement, $\mathcal{A}(\mathcal{M})$, cf. Def. 4.10, of pairs of rankings provided by the similarity measures introduced, and compared to humans. Boxplot diagrams visualize the distribution of the ranking agreement over all queries, including minimum, maximum, and median. The upper and lower boundaries of a box express the upper and lower quartile. We observe that AD and RSD strongly agree on the ranking, which is a specific property of the process model collection. Since it is a reference model, we observed that sets of actions that are common to several models are also executed in the same order, which hints at an operational overlap between these models. Nevertheless, all distances show a certain degree of agreement, indicated by the lower quartile being above 0.5 in Fig. 59a. For all three data series, the median is very close to the lower quartile, and hence can hardly be distinguished in the diagram.

The ranking agreement between each distance measure and the human assessed ranking is presented in Fig. 59b. None of the measures delivers a significantly inferior ranking compared to the human ranking. However, AD aligns best with the human assessment, which can be explained with the intuition that it is easier to identify common actions in a model than comparing its actual structure or behavior. Due to the agreement between AD and RSD, the behavioral similarity aligns well with human assessment in ordering search results.



(a) Pairwise ranking agreement of AD, GED, RSD.

(b) Ranking agreement of AD, GED, RSD with human assessment (HUMAN).

Figure 59: Ranking agreement of search results.

These values have to be considered in a certain context: As the human assessment only uses differentiated values from a Likert scale, the ranking for 85 candidate models can never be unambiguous, as matches with the same Likert value may be randomly ranked and the presented results have been derived from one possible ranking of the human assessed data.

Discussion of Results

For the data gathered from the human assessment of process similarity, we notice that the tendencies of every distance measure resemble those obtained by means of human assessed similarity. Whereas in [75, 158] it has been shown that the application of similarity measures, both structural and behavioral, provides good approximation of human classification of relevant and non-relevant results, we hereby indicate that also the quality and differentiation of the search results produced by similarity measures are well in line with the human perception.

The results support our intuition that quality measures add to the naive ranking produced by the similarity measures, in a way that they give insight into the number of meaningful models, and the quality of the provided ranking. Hence, they improve the prediction of quality of the search results for a user searching models in a repository.

Correlation of the measures

In order to assure that each of our proposed evaluation measures provides expressiveness on its own, we determined the correlation within the measures for confidence and discrimination. A strong correlation between these measures would indicate that at least one of them is redundant. Therefore we assessed the covariance and Pearson correlation coefficient of all pairs of evaluation measures produced in above experiments, listed in Tab. 12.

As the codomain of our measures is $[0, 1]$, we expect small values for the covariance. Thus, the covariance as such does not yet prove linear independence. Additionally we determine the Pearson correlation coefficient which normalizes the values of the covariance. The Pearson correlation values lie between -0.32 and 0.38, which provides no indication for linear dependency of the measures.

MEASURES		COVARIANCE	PEARSON COEFFICIENT
$\mathcal{C}_1$	$\mathcal{C}_{most}$	0.0189	0.3842
$\mathcal{C}_1$	$\mathcal{D}_{most}$	0.0122	0.1983
$\mathcal{C}_1$	$\mathcal{D}_{all}$	-0.0049	-0.1185
$\mathcal{C}_{most}$	$\mathcal{D}_{most}$	-0.0116	-0.3194
$\mathcal{C}_{most}$	$\mathcal{D}_{all}$	0.0052	0.2132
$\mathcal{D}_{most}$	$\mathcal{D}_{all}$	-0.0075	-0.2475

Table 12: Covariance & Pearson correlation coefficient for quality measures.

8

DISCUSSION AND CONCLUSION

In the final chapter, we reflect on the contributions made in this thesis. First, we showcase a software prototype in Sect. 8.1 that comprises a proof-of -concept of the framework of our research and provides implementations for querying and similarity search. Then we summarize the results that have been achieved, in Sect. 8.2. We conclude this chapter with a discussion of limitations of the presented work and how they might be addressed in future research in Sect. 8.3.

8.1 IMPLEMENTATION

Based on the findings of this thesis, we have implemented a process model search platform that allows for the integration of dedicated search engines in a plugin architecture. Hence, it enables employing a diversity of different search methods and techniques in a common framework.

While the design of this platform has been driven by the formal framework presented in Chap. 4, it has a more generic character and supports also techniques that are not grounded on successor relations. To this end, the platform reduces the time required to implement and evaluate a search technique and enables the comparison of different search approaches. In this section, we briefly report on the architecture of and usage scenarios of this system.

Architecture

The system has been devised as a platform for search. However, during the development, it turned out that search and evaluation share many commonalities. Essentially, evaluation takes a set of reference queries and candidates and runs a series of search requests. Quality can be evaluated by comparing the returned search result against reference data and performance by measuring certain indicators during the search. This has been considered in the design of the system illustrated in Fig. 60.

The platform embraces a traditional two-tiered web application, where the interfaces for search and for evaluation have been implemented as HTML5 applications that run completely in a web browser. The backend which comprises the search engines has been implemented in Java. The platform server interacts via a JSON API with the web interface and provides functionality to load process models from

1 Fig. 60 is modeled using FMC [147]; see Appendix A for a brief explanation.

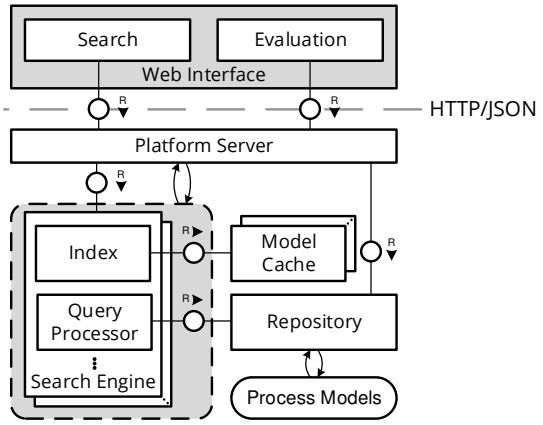


Figure 60: Architecture[1] of the search platform.

the repository and accept search requests. A dedicated search engine comprises at least a query processor that takes a query as input and computes a search result that is ranked by some distance, cf. Sect. 4.3. Additionally, an index can be provided that maintains information required for the efficient retrieval of matches. To facilitate the implementation of these components, the search platform provides shared components, i. e., model caches that underpin a custom index and a persistence interface with the repository that stores and manages process models. The model cache increases startup speed as it preserves data that has been precomputed from models stored in the repository.

The platform can manage a number of search engines at the same time. If more than one engine is deployed to the platform, the user can choose from the available engines when posing a search question. The interface between the platform server and the search engine allow providing a set of configuration parameters that are displayed to the user in the search interface.

Usage Scenarios

Despite the vast number of approaches to process model search, virtually none has been presented as a running system and evaluation has been conducted under lab conditions with a minimal implementation to support query processing. This is, arguably, due to the effort of providing a complete process model search infrastructure that includes a user interface to formulate a query, a process model repository to store, manage, and retrieve models, and the visual presentation of search results as a response to the query. This functionality is shared among all process model search techniques.

Search

The prime focus of our platform was therefore to offer this functionality and provide a set of APIs to integrate dedicated search engines. To this end, we have implemented a user interface as web application that uses a minimal process model editor that allows formulating queries as BPMN process models. We have integrated both techniques presented in this thesis, i. e., *querying process models by example*, cf. Chap. 5 and *searching process models by similarity*, cf. Chap. 6. The model editor itself can be easily extended which allows formulating queries also in other languages, e. g., BPMN-Q [6]. As we resort to jBPT[2] for the computation of successor relations and relation sets, the query is transformed to a Petri net system. The collection of process models, we used for our evaluation has also been transformed accordingly, and hence, we present search results as Petri nets. The presentation of the search results includes a computation of the quality measures presented in Sect. 4.4 and projects successor relations that lead to a match according to Sect. 4.5. A screenshot of the query editor and an exemplary search result are depicted in Fig. 61.

2 http://code.google.com/p/jbpt/

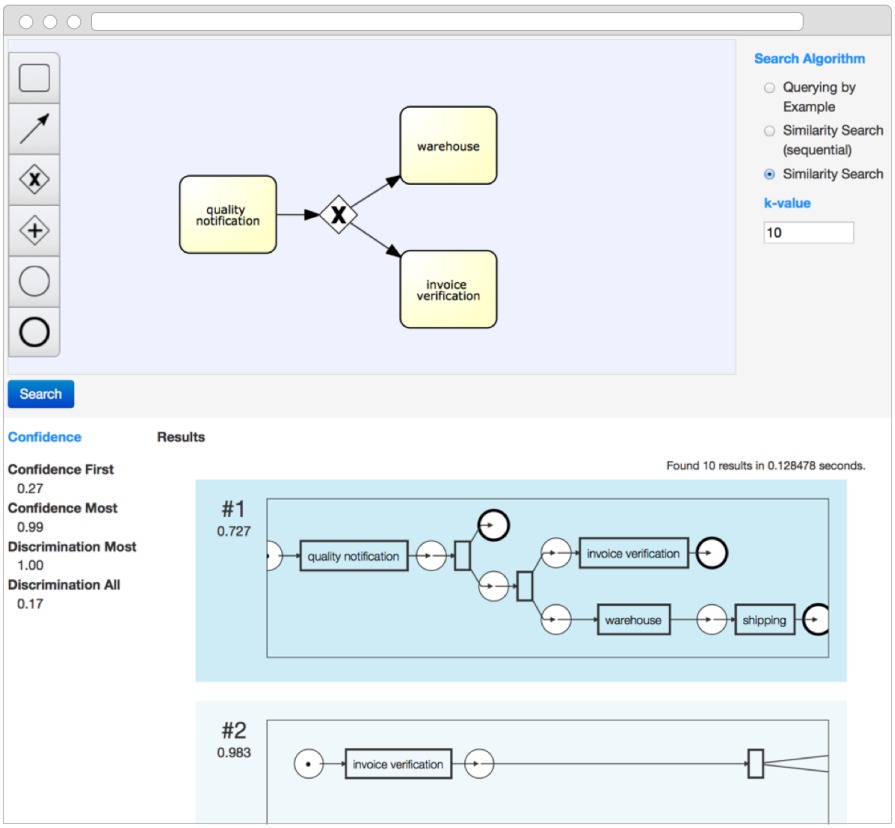


Figure 61: Search interface of process model search platform.

Evaluation

The evaluation web application is a prototype to show that the platform can also be used to evaluate a search approach. Therefore, we have implemented a set of evaluation methods presented in Sect. 7.1 and a sample experiment that includes the data set used for our evaluation of similarity search, cf. Sect. 7.4. For instance, the values for precision and recall are provided in a table for each query and as a precision-recall cluster diagram, which allows for fast identification of queries with superior or inferior quality. For performance evaluations, we devised an API that provides stopwatches and counters that can be employed by the dedicated search engine to measure raw performance data. As these measures are contained in a search result, they can be analyzed by an evaluation application in a straightforward manner.

8.2 RESULTS OF THIS THESIS

Process model search is a prevailing research topic and a plethora of search techniques, both structural and behavioral, has been proposed in the literature. Yet, these techniques emphasize concepts to deciding a match and measuring similarity of a pair of business process models, and fall short in providing a coherent and comprehensive

solution that also addresses efficient search and the presentation of a search result to facilitate reuse of the proposed matches.

As business process models describe the actions and their relations to carry out the operations of a business, they are primarily models of behavior. Hence, we assume a behavioral perspective for process model search. To cope with the exponential growth of sets of traces and state spaces, we rely on successor relations, an abstraction of the behavior of process models that captures the ordering of pairs of actions in process instances. With regard to the goals stated in Sect. 1.2, we have developed a comprehensive search approach that does not require intricate query languages but takes a regular process model as input, to address a broad audience of business process management stakeholders. The approach provides capabilities for efficient querying and similarity search, and offers solutions for an effective presentation of search results. In particular, the following results have been achieved.

Querying Process Models by Example

We presented a novel querying approach for process models, based on the assumption that a query comprises an example process model that contains only few yet relevant actions and their ordering relations. Process models that are able to replay the behavior of the query are considered a match and are presented in a ranked order. Deciding a match is based on abstract trace inclusion, which requires that each trace a query can produce is included in an abstraction of at least one trace of a matching model. Ranking is guided by the closeness of a match to the query, i. e., by the amount of behavioral abstraction of the query. For the class of sound, free-choice workflow systems, successor relations have been proven to decide a match and compute closeness efficiently. We illustrated how these concepts can be used to query process models and support the construction of an index.

Moreover, we evaluated our approach toward its effectiveness by means of a user study, where we focused on the quality of a search result compared to a human assessment of potential matches. In a quantitative experiment, we measured search times for increasing sizes of process model collections and showed that the index provides good scalability in a practical setting.

Searching Process Models by Similarity

In contrast to querying, similarity search assumes a complete documentation of a business process and obtains process models that resemble the query, while some features of the query might remain unmatched. Several behavioral similarity measures have been proposed in literature, of which the majority focuses on comparing exe-

cution traces and few on behavioral relations. In view of these works, our approach to similarity search acknowledges the confidence of a process alignment and employs elementary similarity measures that focus on particular aspects of behavioral relations, i. e., exclusiveness, strict order, and interleaving order of the actions in a process models. The measure can be configured by the look-ahead of the behavioral relations and by individual weights for the elementary similarity measures. We proposed a mechanism to derive these weights from the characteristics of a process model collection and presented an approach to apply the similarity measure to metric space indexes that reduce the number of required comparison operations to carry out search and thus enables efficient search.

In a set of experiments, we validated the quality of the similarity measure and the effectiveness of its automatic configuration, and examined the scalability of search with the metric space index. The experiments revealed good quality of the search techniques and a substantial increase in efficiency compared to exhaustive search.

Quality Measures for Search Results

To assist users in estimating the quality of a search result, we proposed a set of five quality measures that examine statistical aspects of the search result. These measures offer answers to questions how differentiated a search result is, and whether the provided ranking is unambiguous. The agreement of a ranking with alternative rankings, provided, for instance, with a set of dedicated distance functions, can assist in supporting or improving the ranking of a search result. We applied our findings to a reference process model collection and comprehensively evaluated their prediction of quality with respect to a human assessment of process similarity.

Although we resort to business process model repositories and use experiment data of a reference process model collection for our evaluation, these results apply to all use cases that exhibit large collections of complex data to be searched with by means of distances.

Projection of Successor Relations to Process Models

Supporting users in understanding matches is crucial for effective search techniques and facilitates reusing models retrieved by search. To this end, we presented an approach to project a subset of a successor relation that has been identified as commonalities with a query to the graph structure of a process model. After discovering a finite set of firing sequences that correspond to successor relation pairs, we show how to map them to a process model graph. While this requires the analysis of the state space of process models in the general case, we provide an efficient projection based purely on behavioral relations that can be computed efficiently.

Since both search techniques introduced in this thesis are based on common successor relations, the projection of matched relation pairs can be applied to querying and similarity search results, likewise.

Behavioral Distances for Process Model Search

Each of the above results is grounded on a common formal framework that establishes process model search by distances that quantify behavioral commonalities between a query and a candidate: Search techniques — querying and similarity search — provide dedicated distance functions that decide on a match by means of an overlap of their successor relations or relation sets. A ranking of search results is based on the notion of a non-negative, monotonic distance function and quality measures employ the distances exposed by a ranking to judge on search results. Finally, the projection of matched behavior relies on the common set of successor relation pairs, as it is provided by the search techniques. We have reported on an implemented proof-of-concept that shows the applicability and the suitability of the proposed concepts in practice.

The findings of this thesis are applicable in a broader context. For instance, any distance measure that takes a pair of business process models as input and satisfies above properties can be applied to the framework in general, even if these techniques do not rely on successor relations. Search can be conducted as range or nearest neighbor search, and quality measures for the search result apply. We have further outlined that the projection of successor relations to the process model offers advantages in other fields of model management, e. g., the automatic construction of models from templates and process model refactoring.

8.3 LIMITATIONS AND FUTURE RESEARCH

The work presented on this thesis is based on a number of assumptions and limitations. We briefly address these and outline possible venues to solve them in future research.

Petri Net Systems

The contributions of this thesis are formally grounded on the notion of Petri net systems, and a particular subclass, namely sound, free-choice workflow systems. We argued that the majority of correct business process models can be transformed to this class of net systems. However, there are undoubtedly issues with this transformation that yield from certain behavioral patterns [290]. For example, multi-instance, a behavioral concept that allows the execution of a dynamic number of instances of an action concurrently, or the in-

clusive join cannot be represented in Petri nets precisely. Moreover, non-procedural process modeling languages are also not covered.

The use of Petri nets in our work does not impose a severe restriction, as successor relations are defined for traces a business process can produce. We rely on net systems rather as a vehicle to show efficient algorithms for the analysis and visualization of business processes that applies to sound, free-choice net systems.

Nevertheless, traces can be produced in each of the above situations, which allows deriving successor relations and, following, applying our techniques to process model search. For instance, for advanced behavioral patterns of BPMN, an execution-compliant process engine could compute the set of traces by simulation. This step would be contained in the *feature extraction* stage of the process model search lifecycle, cf. Sect. 2.4. Furthermore, efficient approaches to compute successor relations directly from business process models with precise semantics may be devised in future.

Granularity of Process Models

Our research relies on process alignments that comprise corresponding features of pairs of process models, cf. Sect. 3.4. In this thesis, we resorted to elementary alignments of the actions of business process models, i. e., an action of one process model can be corresponding with at most one action of a second process model. In practice, however, this assumption may not hold. Complex correspondences, presented in [316], are able to capture different levels of granularity by relating sets of actions of one model with sets of actions of another model. While complex correspondences suggest an improvement over elementary ones in view of differences in granularity, the implications of matching process models with the notion of complex correspondences are not straightforward.

Consider the example in Fig. 62 that illustrates the alignment between three net systems. Here, transition A of Fig. 62a corresponds with transitions A_1 and A_2 of Fig. 62b. Assume that A refers to sending a customer an invoice via email and letter, the former for a swift notification and the latter as a legally binding document. If A_1 refers to sending an email and A_2 to a letter, then the customer will only receive one of these messages and miss the other. Hence, there exists a behavioral difference between the corresponding actions. An analogue case is modeled with transitions B_1 and B_2 in Fig. 62a and transition B in Fig. 62b. Hence, different operational semantics of actions shall be acknowledged by behavioral semantics in the model.

Moreover, some aspects of granularity may not be captured by complex correspondences. Referring to Fig. 62c, action B can be repeated, if the outcome of one iteration is not satisfactory, which is represented in the successor relation, i. e., $(\mathsf{B}, \mathsf{B}) \in >^{S}_{b(S)}$. The same behavior may be comprised by transition B in Fig. 62b, yet the iterative semantics

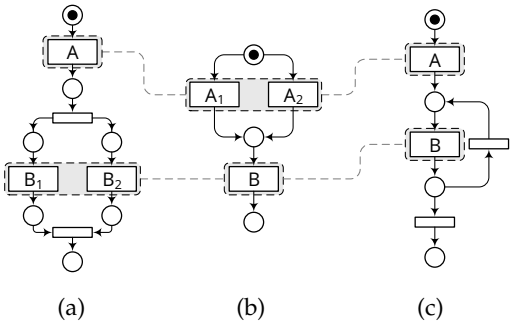


Figure 62: An alignment with complex correspondences.

have not been modeled explicitly, but are contained in the semantics of the action represented by transition B. Hence, the commonalities of this correspondence would be missed by the complex alignment, as the presented case requires relating fragments of process models.

Therefore, we argue that addressing different levels of granularity shall incorporate the semantics of each action in the process model into the analysis of behavior. An analysis of the semantics of an action is, however, beyond the scope of this thesis. Due to implicit assumptions of the process designer and the limited linguistic context provided by process models, this is far from trivial, and shall be addressed in future work.

Approximate Querying

We have defined process model querying by means of abstract trace inclusion, that is, each trace of the query must correspond to a trace of the match, once a certain amount of transition occurrences has been abstracted from the match's trace. As a consequence, the behavior of a query must be completely contained in a match. Our experiments indicated that this may be a requirement too strict, in particular with regard to parallel splits, or more precisely, concurrent actions.

Concurrent actions in a query business process model are transformed into a pair of transitions in the corresponding net system that can be enabled concurrently, which leads to an interleaved ordering represented in the traces of that system and symmetric relation pairs in the successor relation. Consequently, any candidate that matches this behavior must provide traces that provide the interleaved ordering as well. This has been acknowledged only by few experts in the human assessment of matches in our experiment, cf. Sect. 7.3, which suggests that also partial matching of a query model is worth to be followed upon.

This issue can be approached from a number of different angles. For instance, instead of requiring the inclusion of the query successor relation in the relation of a candidate as a necessary condition, cf. Thm. 5.2, one could accept a candidate, if at least one relation pair

is matched. The number of matched relation pairs would then determine the quality of a match. Another solution is the introduction of new behavioral relations and a different treatment of the semantics expressed by a query model. In [318], a *co-occurrence relation* has been introduced, which denotes for a pair of transitions that if either transition occurs in a trace the other must appear as well, whereas their order may be arbitrary. Then, concurrent transitions of a query might not be considered for symmetric successor relation pairs, but treated merely as a co-occurrence. Consequently, any candidate that allows executing the corresponding transitions in any order and ensures that they are executed together would be taken into account for a match.

However, any of the solutions to relaxed querying semantics, which we refer to as approximate querying, requires further investigation toward human assessment, when searching by means of behavioral inclusion. This includes experiments, where users formulate queries themselves to express a given search question in order to evaluate their expectations.

Searching the Information Dimension

In our work, we resort to the behavior of process models in terms of successor relations that yield from the control flow of business process models. However, as we explained in Sect. 2.2, process models expose a number of dimensions. In particular the information dimension has seen considerable recognition in business process management research, lately. This is due to the importance of information, be it data or physical goods, that serves as the input of a process, is processed during its enactment, and is created as a process result.

Figure 63 shows a simplified order fulfillment process with a number of data objects. Moreover, a data object might change its state during the advancing process, for instance, after an order is received, it becomes verified. After issuing the shipment, the state of the order becomes shipped, and after the payment of the customer has been received it is fulfilled.

State changes of data objects can be treated as events. At the same time, successor relations also express the ordering of events, since the firing of a transition with business semantics is understood as

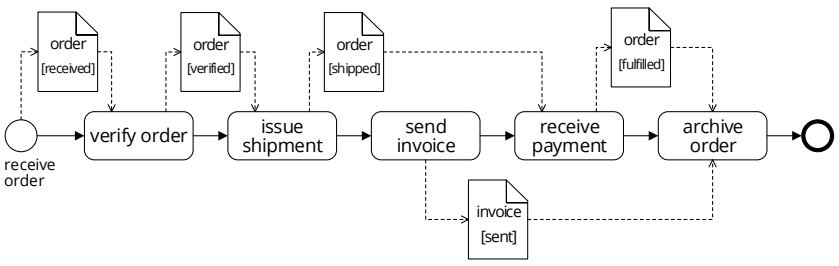


Figure 63: Data flow in the business process.

the termination of the corresponding action, cf. Sect. 4.1. Hence, it is possible to derive successor relations also for the state transitions of data objects. Consequently, the provided framework and search techniques for the behavior of business processes can be applied to searching for commonalities in the lifecycle of data objects.

Part IV

APPENDIX

A

FUNDAMENTAL MODELING CONCEPTS

The Fundamental Modeling Concepts (FMC) are a modeling notation and method to capture and exchange knowledge about software-intensive systems [147]. In particular, block diagrams enable the crisp yet effective visualization of software architectures on various levels of abstraction, whereas the focus is on active and passive components that interact with each other.

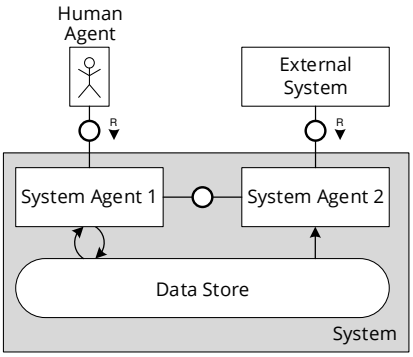


Figure 64: Example diagram showing the Fundamental Modeling Concepts notation for block diagrams.

The notation for block diagrams consists of two types of entities; an example is shown in Fig. 64. *Agents*, depicted by rectangles, are active components of a system that serve a well-defined purpose. Agents can either be systems or human agents, whereas system agents can be hierarchically decomposed. Human agents are pointed out by a stick man figure. *Locations* are depicted by circles, ellipses, or rectangles with rounded corners and represent passive system components, i.e., storage or channels. Whereas a storage is used by one or more agents to store some data, a channel represents communication between a pair of agents.

Locations and agents are connected by arcs. Passive components cannot interact with each other directly, but provide the medium for agents to communicate. Therefore, FMC models are bipartite. The direction of an arrow indicates the direction in which data flows. Data flow of channels can be undirected, which indicates that the direction of data interchange is not restricted. An arrowhead with the annotation `R` denotes that the communication between the involved agents is of the request-reply-pattern, i.e., the agent to which the arrowhead points does not contact the other agent directly, but only responds to their requests.

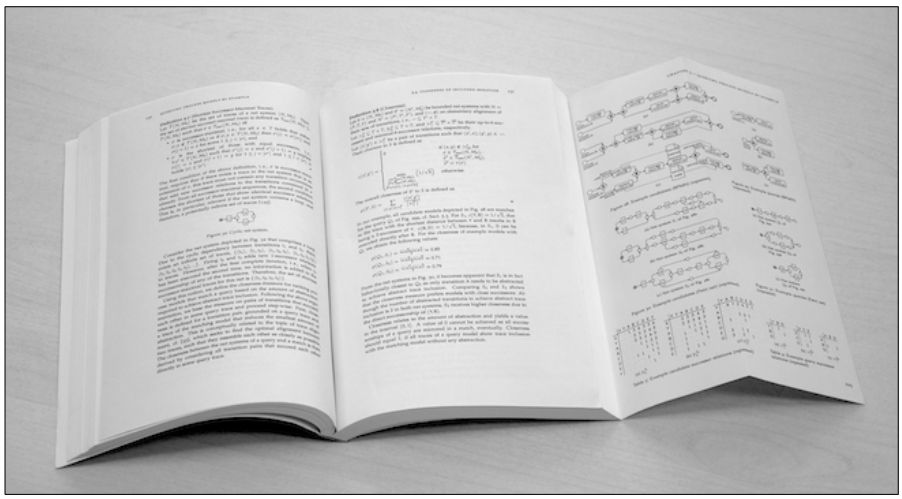
Figure 64 shows a system that comprises two agents and a data store. A human agent and an external system interact with this system. The human agent interacts with the system through System Agent 1. Typically, humans interact with computer systems using the request-reply-pattern, as the user takes initiative in the interaction. However, sometimes the system has to prompt the user, e. g., to inform them about an event. In contrast, System Agent 2 is responsible to handle requests from the External System. System Agents 1 and 2 can communicate arbitrarily with each other. While System Agent 1 can read and modify data stored in the Data Storage, System Agent 2 can only read the data.

B

RUNNING EXAMPLES

Each chapter of Part II of this thesis introduces a running example by which methods and results are explained in a coherent fashion. These examples are developed throughout the chapter. The business process models, modeled in BPMN, their respective Petri net systems, and matrices of their successor relations and relations sets, respectively, are repeated on the following pages.

In the printed version of this thesis, these pages can be folded out in a fashion that they accompany the reader throughout the lecture of each chapter and save them the effort to thumb through the document to recall these figures.



Running examples can be folded out for reference during the lecture of each chapter.

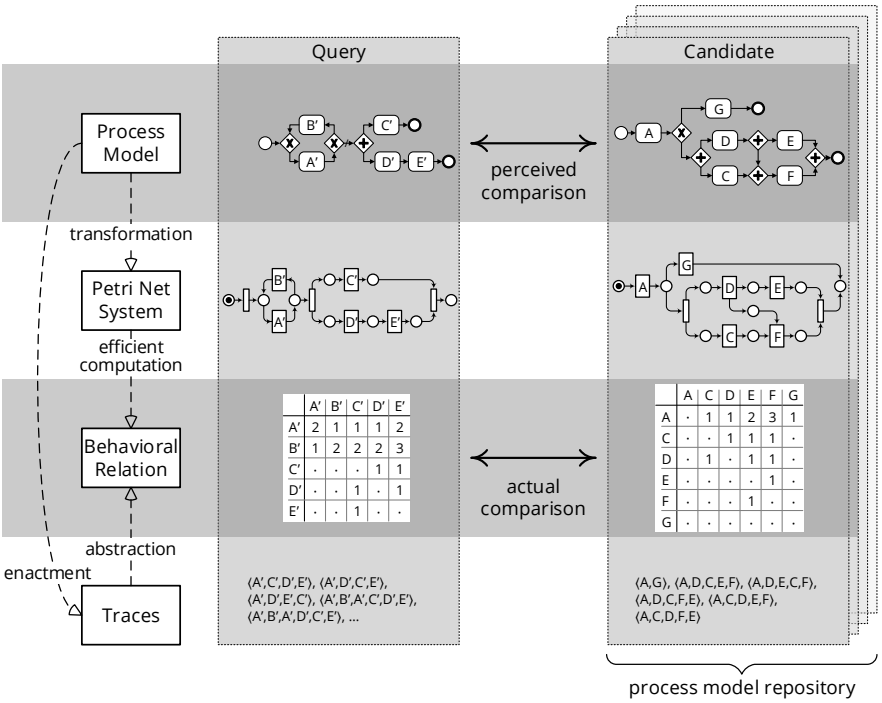
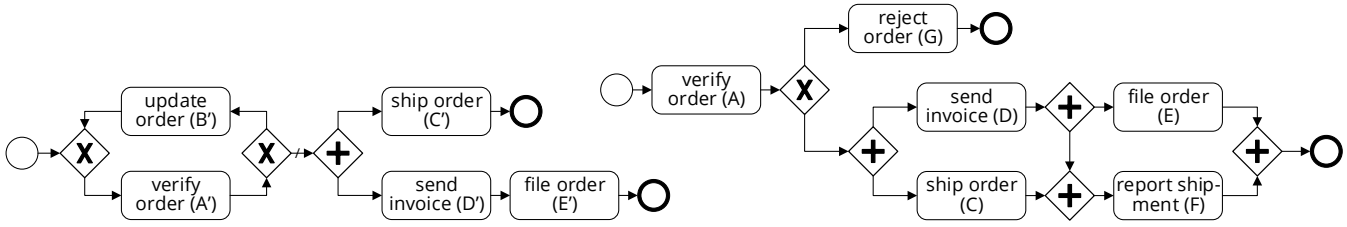
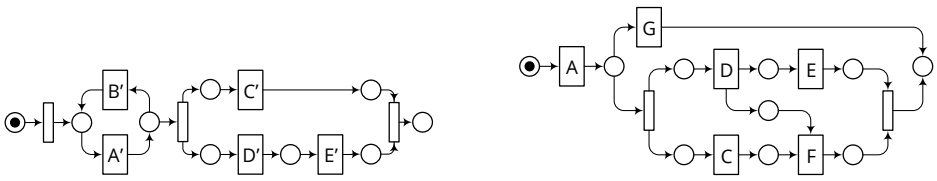


Figure 14: Framework outline (repeated).



(a) Order handling process query. (b) Order handling process candidate.

Figure 15: Order handling (BPMN) (repeated).



(a) Order handling process query. (b) Order handling process candidate.

Figure 16: Order handling (Petri net) (repeated).

	A′	B′	C′	D′	E′
A′	2	1	1	1	2
B′	1	2	2	2	3
C′	·	·	·	1	1
D′	·	·	1	·	1
E′	·	·	1	·	·

(a) $>_k^{S'}$ of query net system in Fig. 16a.

	A	C	D	E	F	G
A	·	1	1	2	3	1
C	·	·	1	1	1	·
D	·	1	·	1	1	·
E	·	·	·	·	1	·
F	·	·	·	1	·	·
G	·	·	·	·	·	·

(b) $>_k^{S}$ of candidate net system in Fig. 16b.

Table 2: Order handling successor relations (repeated).

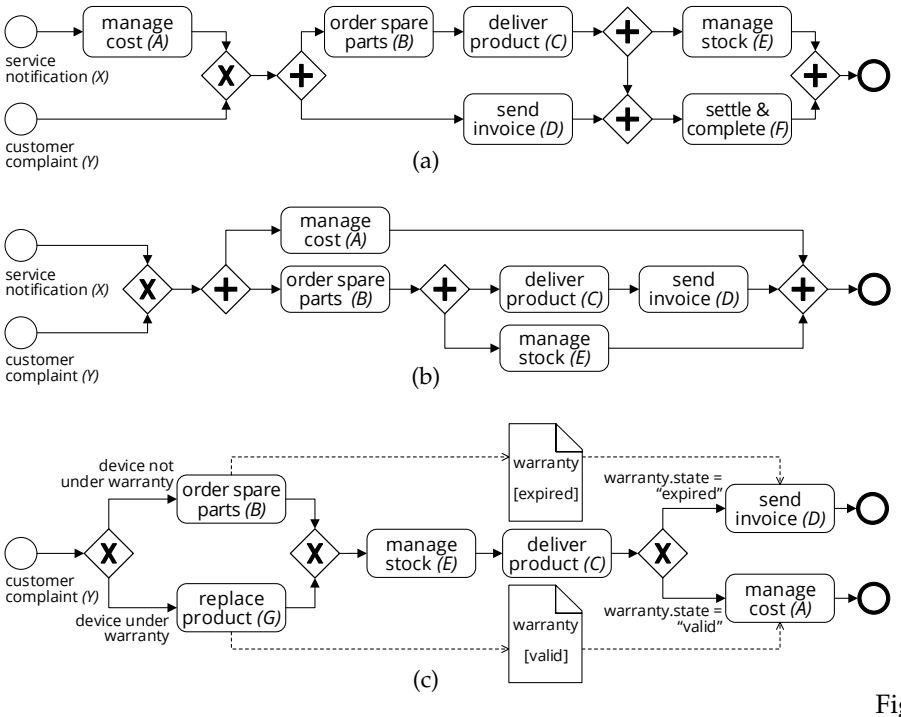


Figure 28: Example candidates (BPMN) (repeated).

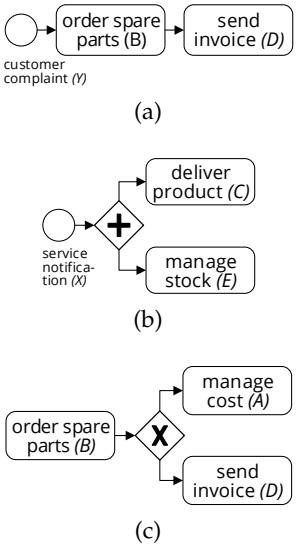
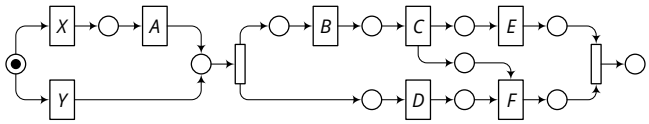
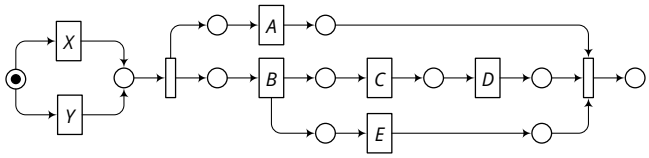


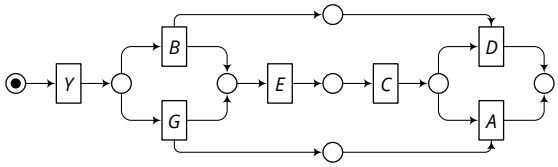
Figure 29: Example queries (BPMN) (repeated).



(a) Net system S_1 of Fig. 28a.



(b) Net system S_2 of Fig. 28b.



(c) Net system S_3 of Fig. 28c.

Figure 30: Example candidates (Petri net) (repeated).

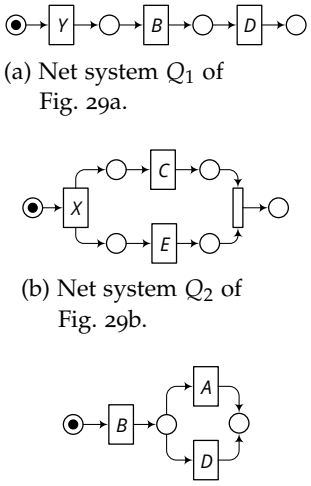


Figure 31: Example queries (Petri net) (repeated).

	X	Y	A	B	C	D	E	F
X	·	·	1	3	4	3	5	6
Y	·	·	·	2	3	2	4	5
A	·	·	·	2	3	2	4	5
B	·	·	·	·	1	1	2	2
C	·	·	·	·	·	1	1	1
D	·	·	·	1	1	·	1	1
E	·	·	·	·	·	1	·	1
F	·	·	·	·	·	·	1	·

(a) $\trianglerighteq_k^{S_1}$

	X	Y	A	B	C	D	E
X	·	·	2	2	3	4	3
Y	·	·	2	2	3	4	3
A	·	·	·	1	1	1	1
B	·	·	1	·	1	2	1
C	·	·	1	·	·	1	1
D	·	·	1	·	·	·	1
E	·	·	1	·	1	1	·

(b) $\trianglerighteq_k^{S_2}$

	Y	A	B	C	D	E	G
Y	·	4	1	3	4	2	1
A	·	·	·	·	·	·	·
B	·	·	·	2	3	1	·
C	·	1	·	·	1	·	·
D	·	·	·	·	·	·	·
E	·	2	·	1	2	·	·
G	·	3	·	2	·	1	·

(c) $\trianglerighteq_k^{S_3}$

Table 3: Example candidate successor relations (repeated).

	Y	B	D
Y	·	1	·
B	·	·	1
D	·	·	·

(a) $>_1^{Q_1}$

	X	C	E
X	·	1	1
C	·	·	1
E	·	1	·

(b) $>_1^{Q_2}$

	A	B	D
A	·	·	·
B	1	·	1
D	·	·	·

(c) $>_1^{Q_3}$

Table 4: Example query successor relations (repeated).

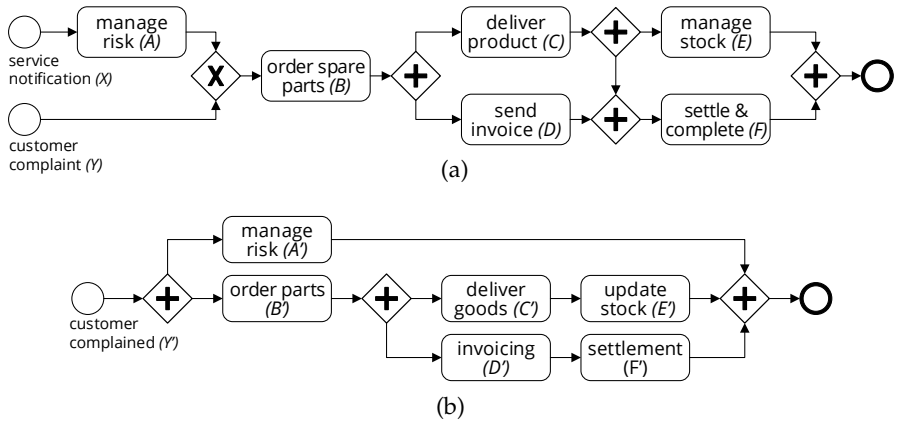
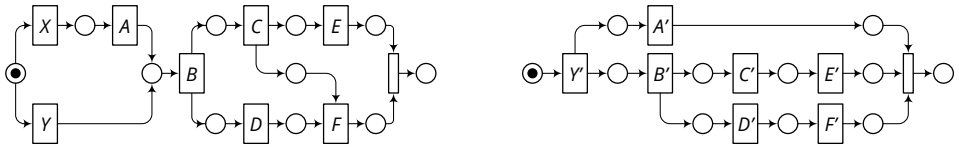


Figure 34: Example processes (BPMN) (repeated).



(a) Net system S. (b) Net system S'.

Figure 35: Example processes (Petri net) (repeated).

	X	Y	A	B	C	D	E	F
X	·	·	1	2	3	3	4	4
Y	·	·	·	1	2	2	3	3
A	·	·	·	1	2	2	3	3
B	·	·	·	·	1	1	2	2
C	·	·	·	·	·	1	1	1
D	·	·	·	·	1	·	1	1
E	·	·	·	·	·	1	·	1
F	·	·	·	·	·	·	1	·

(a) $>_k^S$ of net system in Fig. 35a.

	X	Y	A	B	C	D	E	F
X	+	+	→	+	+	+	+	+
Y	+	+	+	→	+	+	+	+
A	←	+	+	→	+	+	+	+
B	+	←	←	+	→	→	+	+
C	+	+	+	←	+	‖	→	→
D	+	+	+	←	‖	+	‖	→
E	+	+	+	+	←	‖	+	‖
F	+	+	+	+	←	←	‖	+

(a) $\mathcal{B}_1^S$ of net system in Fig. 35a.

	X	Y	A	B	C	D	E	F
X	+	+	→	→	→	→	→	→
Y	+	+	+	→	→	→	→	→
A	←	+	+	→	→	→	→	→
B	←	←	←	+	→	→	→	→
C	←	←	←	←	+	‖	→	→
D	←	←	←	←	‖	+	‖	→
E	←	←	←	←	←	‖	+	‖
F	←	←	←	←	←	←	‖	+

(a) $\mathcal{B}_k^S$ of net system in Fig. 35a.

	Y'	A'	B'	C'	D'	E'	F'
Y'	·	1	1	2	2	3	3
A'	·	·	1	1	1	1	1
B'	·	1	·	1	1	2	2
C'	·	1	·	·	1	1	1
D'	·	1	·	1	·	1	1
E'	·	1	·	·	1	·	1
F'	·	1	·	1	·	1	·

(b) $>_k^{S'}$ of net system in Fig. 35b.

Table 7: Example successor relations (repeated).

	Y'	A'	B'	C'	D'	E'	F'
Y'	+	→	→	+	+	+	+
A'	←	+	‖	‖	‖	‖	‖
B'	←	‖	+	→	→	+	+
C'	+	‖	←	+	‖	→	‖
D'	+	‖	←	‖	+	‖	→
E'	+	‖	+	←	‖	+	‖
F'	+	‖	+	‖	←	‖	+

(b) $\mathcal{B}_1^{S'}$ of net system in Fig. 35b.

Table 9: Example relation sets ($k = 1$) (repeated).

	Y'	A'	B'	C'	D'	E'	F'
Y'	+	→	→	→	→	→	→
A'	←	+	‖	‖	‖	‖	‖
B'	←	‖	+	→	→	→	→
C'	←	‖	←	+	‖	→	‖
D'	←	‖	←	‖	+	‖	→
E'	←	‖	←	←	‖	+	‖
F'	←	‖	←	‖	←	‖	+

(b) $\mathcal{B}_k^{S'}$ of net system in Fig. 35b.

Table 10: Example relation sets ($k = b(S)$) (repeated).

C

GLOSSARY

$\mathcal{P}(S)$	Powerset, i.e., the set of all subsets of a set S (Not. 3.2, page 44)
σ	Sequence (Not. 3.3, page 45)
$;$	Concatenation of sequences (Not. 3.3, page 45)
N	Petri net $N = (P, T, F)$ (Def. 3.1, page 45)
P	Set of places of a net (Def. 3.1, page 45)
T	Set of transitions of a net (Def. 3.1, page 45)
F	Flow relation of a net (Def. 3.1, page 45)
$\bullet t$	Preset of a transition t (page 45)
$t\bullet$	Postset of a transition t (page 45)
(N, M_0)	A net system (Def. 3.2, page 46)
$\mathbb{M}$	Set of all markings of a net system (Def. 3.2, page 46)
σ	Firing sequence, trace of a net system (page 47)
$\mathcal{T}$	Set of all traces of a net system (Def. 3.4, page 47)
$\perp$	Transition of a net system that has no label assigned are referred to by $\perp$
$\rhd_k^S$	k-successor of a sequence $\rhd_k^\sigma$, of a system $\rhd_k^S$ (Def. 3.9, page 51)
$>_k$	Up-to-k-successor of a sequence $>_k^\sigma$, of a system $>_k^S$ (Def. 3.10 (page 52)
$\trianglerighteq_k$	minimal k-successor of a sequence $\trianglerighteq_k^\sigma$, of a system $\trianglerighteq_k^S$ (Def. 3.11, page 53)
$\|\|_c^S$	Concurrency relation (Def. 3.12, page 53)
$\backsim$	Correspondence Relation (Def. 3.13, page 56)
ϕ	Confidence Function (Def. 3.13, page 56)
$(\backsim, \phi)$	Alignment of two nets (Def. 3.13, page 56)
$\mathcal{R}$	The universe of behavioral relations $R, R' \in \mathcal{R}$ (Def. 4.1, page 80)
$\breve{\cap}$	Intersection set of successor relations by a given alignment (Def. 4.3, page 82)
$p(R', R)$	Precision of process behavior (Def. 4.4, page 83)
$r(R', R)$	Recall of process behavior (Def. 4.4, page 83)

d_f	Behavioral distance measure based on the f-score (page 85)
Ω	Process model search result (Def. 4.5, page 86)
$r(R')$	Search radius of a query (Not. 4.1, page 87; Def. 5.1, page 108)
$min(\Omega)$	Minimum distance of a search result (Not. 4.2, page 89)
$med(\Omega)$	Median distance of a search result (Not. 4.2, page 89)
$max(\Omega)$	Maximum distance of a search result (Not. 4.2, page 89)
$\mathcal{C}_1(\Omega)$	Confidence of the best match of a search result (Def. 4.6, page 90)
$\mathcal{C}_{most}(\Omega)$	Confidence of the superior partition of a search result (Def. 4.7, page 91)
$\mathcal{D}_{most}(\Omega)$	Discrimination of the superior partition of a search result (Def. 4.8, page 92)
$\mathcal{D}_{all}(\Omega)$	Discrimination of the complete search result (Def. 4.9, page 93)
rank	Partial function, $rank : \mathcal{R} \times \Omega^v \nrightarrow \mathbb{N}$, that returns the ranking position of a match in the search result (page 94)
diff	Ranking difference function (Def. 4.10, page 94)
$\mathcal{A}(\mathcal{M})$	Ranking agreement of a set of rankings $\mathcal{M} = \{\Omega^1, ..., \Omega^m\}$ (Def. 4.10, page 94)
$\gg_1(x, y)$	Successor decomposition of a transition pair (x, y) in successor relation (Def. 4.13, page 103)
$\mathcal{C}$	The set of all possible process models (Def. 5.1, page 108; Def. 6.1, page 142)
$\mathcal{Q}$	The set of all possible queries (Def. 5.1, page 108)
$\mathcal{F}$	The set of all possible features of process models or queries (Def. 5.1, page 108)
$\overline{T}$	Projection of transitions, firing sequences, and traces, such that $\overline{T} \subseteq T$ (Def. 5.2, page 120)
τ	Loose abstraction of a firing sequence (Def. 5.3, page 121)
$\backsimeq$	Equality that acknowledges the correspondence relations $\backsim$ of a given alignment in comparing firing sequences (Def. 5.4, page 121)
$\overset{\sim}{\sqsubseteq}$	Abstract trace inclusion (Def. 5.5, page 122)

$\overset{\sim}{\subseteq}$	Successor inclusion (Def. 5.6, page 124)
$\mathcal{T}_{max}(N, M_0)$	Shortest successor-maximal traces of a net system (Def. 5.7, page 130)
$\rho(S', S)$	Closeness of two net systems S' and S (Def. 5.8, page 131)
$\rho_S(S', S)$	Successor closeness of two net systems S' and S (Def. 5.9, page 132)
$\rho_0(S', S)$	Trivial closeness of two net systems S' and S for queries that comprise one action (Def. 5.10, page 134)
$sim_>(>_k^{S'}, >_k^S)$	Naive up-to-k-successor relation similarity (Def. 6.2, page 156)
$\Phi(X)$	Aggregated confidence score over a set of corresponding relation pairs $X \subseteq \backsim \times \backsim$(Def. 6.3, page 158)
$sim_\Phi(>_k^{S'}, >_k^S)$	Probabilistic up-to-k-successor relation similarity (Def. 6.4, page 158)
$ldn(a, b)$	Normalized string edit similarity of two strings a and b (Not. 6.1, page 159)
$\mathcal{B}_k$	Relation set (Def. 6.5, page 160)
$+_k$	Exclusiveness relation of a relation set (Def. 6.5, page 160)
$\rightarrow_k$	Order relation of a relation ret (Def. 6.5, page 160)
$\leftarrow_k$	Inverse order relation of a relation set (Def. 6.5, page 160)
$\|\|_k$	Interleaving order relation of a relation set (Def. 6.5, page 160)
$\mathcal{B}$	Universe of relation sets (Def. 6.6, Def. 6.7, Def. 6.8, page 162f)
$sim_i(\mathcal{B}_k^{S'}, \mathcal{B}_k^S)$	Elementary relation set similarities, $i \in +, \rightarrow, \|\|$ (Def. 6.6, page 162)
$sim_\mathcal{B}(\mathcal{B}_k^{S'}, \mathcal{B}_k^S)$ $d_\mathcal{B}(\mathcal{B}_k^{S'}, \mathcal{B}_k^S)$	k-relation set similarity and distance (Def. 6.7, page 164)
θ	Alignment threshold (Sect. 7.2, page 186)

D

OVERVIEW OF RELATED WORK

In the following summarize related work on process model querying and similarity search. The approaches are listed in the order of their appearance in the related work sections of this thesis, and provide the following information.

REFERENCE authors and references that present a particular technique

DIMENSION addresses the dimension which is considered by the presented approach, i.e., keywords, structure, or behavior, see Sect. 2.4.

CLASSIFICATION provides a more fine-granular classification of the approaches. The presented classes are discussed in the related work sections, Sect. 5.2 and Sect. 6.2, respectively.

APPROACH describes the underlying concepts or names the provided approach (in case of query languages).

PROCESS ALIGNMENT indicates whether the approach considers an alignment or neglects it altogether, e.g., by treating actions of two models as overlapping sets.

INDEX denotes whether the authors propose algorithms or data structures to make search efficient.

EVALUATION reports on the evaluation of the respective technique. With regards to quality, this requires a comparison with human judgement, as opposed to the presentation of an example and its application to the technique. The listed categories are explained in Sect. 7.1.

REFERENCE	DIMENSION	CLASSIFICATION	APPROACH	PROCESS ALIGNMENT	INDEX	EVALUATION: QUALITY	EVALUATION: PERFORMANCE
Chiu et al. [48]	keywords	text	keyword matching	–	–	effectiveness	query time
Klein et al. [146]	keywords	ontology annotation	annotation matching	–	–	effectiveness	–
Koschmider et al. [150]	keywords	ontology annotation	annotation matching	–	–	–	–
Shao et al. [175, 259]	keywords	text	matching text with labels	–	–	effectiveness	query time
Belhajjame and Brambilla [26]	structure	graph homomorphism	graph homomorphism	✓	–	–	–
Jin et al. [133]	structure	graph homomorphism	containment of paths	–	✓	–	query time, storage size
Pankratius and Stucky [229]	structure	graph homomorphism	Petri net algebra	–	–	–	–
Uba et al. [275] and Ekanayake et al. [82]	structure	graph homomorphism	detection if identical fragments	–	✓	✓	–
Lu et al. [180, 178, 179]	structure	graph homomorphism	homomorphism of depdency graph	–	–	–	–

Table 13: Overview of related work: Querying

REFERENCE	DIMENSION	CLASSIFICATION	APPROACH	PROCESS ALIGNMENT	INDEX	EVALUATION: QUALITY	EVALUATION: PERFORMANCE
Grigori et al. [105, 104]	structure	graph homomorphism	subgraph edit distance	✓	–	effectiveness	query time
Xiao et al. [326]	structure	grph. query language	PNQL	–	–	–	–
Müller [217]	structure	grph. query language	PPML	✓	–	–	–
Beeri et al. in [23, 25] and Deutch and Milo [71]	structure	grph. query language	BP-QL	–	–	–	–
Awad et al. [8, 10, 6]	structure	grph. query language	BPMN-Q	–	✓	–	query time
Yan et al. [330]	structure	grph. query language	Feature nets	–	✓	–	query time
Christophides et al. [51]	structure	text. query language	XML Algebra	–	–	–	–
Choi et al. [49]	structure	text. query language	IPM-QL	–	–	–	–
Missikoff et al. [214, 265]	structure	text. query language	QuBPAL	✓	–	–	–

Table 13: Overview of related work: Querying

REFERENCE	DIMENSION	CLASSIFICATION	APPROACH	PROCESS ALIGNMENT	INDEX	EVALUATION: QUALITY	EVALUATION: PERFORMANCE
Ibanez et al. [129]	structure	text. query language	SPARQL	–	–	–	–
Markovic et al. [193, 191]	structure	text. query language	WSML	✓	–	–	–
Shen and Su [261]	behavior	relations, query language	temporal logic queries	–	–	–	scalability
Song et al. [268]	behavior	relations, query language	LTL queries, restricted to safe PN	–	–	–	query time, storage size
Jin et al. [135]	behavior	relations, query language	composition of behavioral relations	✓	–	–	query time, storage size
Ouyang et al. [228]	behavior	relations, query language	composition behavioral relations	✓	–	–	–
Mahleko et al. [184, 323]	behavior	sequences	containment of n-grams	–	✓	–	scalability
Deutch and Milo [69, 70]	behavior	sequences	containment of sequences	–	–	–	–

Table 13: Overview of related work: Querying

REFERENCE	DIMENSION	CLASSIFICATION	APPROACH	PROCESS ALIGNMENT	INDEX	EVALUATION: QUALITY	EVALUATION: PERFORMANCE
Madhusudan et al. [183]	structure	feature	similarity flooding	–	–	–	–
Humm and Fengel [128]	structure	feature	common model features	–	–	–	–
Akkiraju and Ivan [3]	structure	feature	common model features	–	–	–	–
Ehrig et al. [81]	structure	feature	common actions and their similarity	✓	–	–	–
De Medeiros et al. [66, 293]	structure	feature	common edges	–	–	–	–
Bergmann and Gil [28]	structure	feature	common actions, common edges	–	–	–	–
Gao et al. [94, 95]	structure	feature	aggregate similarity of feature types	✓	–	–	–
Melcher and Seese [200]	structure	feature	similar process model metrics	–	–	–	–
Hornung et al. [126]	structure	vectors	annotations	–	–	✓	–

Table 14: Overview of related work: Similarity search

REFERENCE	DIMENSION	CLASSIFICATION	APPROACH	PROCESS ALIGNMENT	INDEX	EVALUATION: QUALITY	EVALUATION: PERFORMANCE
Jung et al. [137, 138, 139]	structure	vectors	actions and paths	–	–	–	–
Esgin and Senkul [83]	structure	vectors	actions, predecessor and successor	–	–	✓	–
Bae et al. [13]	structure	vectors	block decomposition	–	–	–	–
Bae et al. [15, 14]	structure	vectors	incidence matrix	–	–	–	comparison time
Cuzzocrea et al. [61]	structure	subgraph	largest common subgraph	✓	–	–	–
Niemann et al. [223]	structure	subgraph	sets of common subgraphs	✓	–	effectivness	–
Jin et al. [134]	structure	subgraph	common paths	–	–	–	query time, storage size
Yan et al. [329, 328]	structure	subgraph	characteristic fragments	✓	✓	effectiveness	query time
Dijkman et al. [75, 73, 74]	structure	edit distance	graph edit distance	✓	–	effectiveness	query time
Minor et al. [213]	structure	edit distance	graph edit distance	–	–	effectiveness	–

Table 14: Overview of related work: Similarity search

REFERENCE	DIMENSION	CLASSIFICATION	APPROACH	PROCESS ALIGNMENT	INDEX	EVALUATION: QUALITY	EVALUATION: PERFORMANCE
Küster et al. [161]	structure	edit distance	computation of a changelog	✓	–	–	–
Li et al. [169]	structure	edit distance	high-level edit operations	–	–	–	–
Anon et al. [325, 184, 170, 190]	behavior	sequence	common firing sequence	–	–	–	–
De Medeiros et al. [66, 293]	behavior	sequence	common traces	–	–	–	–
Zha et al. [334]	behavior	sequence	common traces	–	–	–	–
Li et al. [170]	behavior	sequence	common traces	–	–	–	–
Becker et al. [21]	behavior	sequence	common traces	–	–	–	–
Gerke et al. [97]	behavior	sequence	common firing sequence	–	–	–	query time
Wang et al. [307]	behavior	sequence	common principal transition sequences	–	–	–	query time
Sun [271]	behavior	edit distance	string edit distance over sequence	✓	–	–	–
Wombacher et al. [324, 323]	behavior	edit distance	string edit distance over n-grams	–	✓	effectiveness	–

Table 14: Overview of related work: Similarity search

REFERENCE	DIMENSION	CLASSIFICATION	APPROACH	PROCESS ALIGNMENT	INDEX	EVALUATION: QUALITY	EVALUATION: PERFORMANCE
Dongen et al. [75, 300, 205]	behavior	relations	causal footprints	–	–	effectiveness	query time
Zha et al. [334]	behavior	relations	precedence relation	–	–	–	query time
Eshuis and Grefen [84]	behavior	relations	behavioral profiles	–	–	–	–
Kunze et al. [158, 159]	behavior	relations	behavioral profiles	–	✓	effectiveness*	–

Table 14: Overview of related work: Similarity search

BIBLIOGRAPHY

[1] Fabio Aiolli, Andrea Burattin, and Alessandro Sperduti. A Business Process Metric Based on the Alpha Algorithm Relations. In *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 141–146. Springer, September 2011. (Cited on pages 68, 151, 152, 153, and 154.)

[2] Fabio Aiolli, Andrea Burattin, and Alessandro Sperduti. Metric for Clustering Business Processes Based on Alpha Algorithm Relations. Technical report, University of Padua, 2011. (Cited on pages 144, 151, 153, and 154.)

[3] Rama Akkiraju and Anca Ivan. Discovering Business Process Similarities: An Empirical Study with SAP Best Practice Business Processes. In *Service-Oriented Computing*, volume 6470 of *Lecture Notes in Computer Science*, pages 515–526. Springer, 2010. (Cited on pages 29, 31, 32, 112, 147, and 237.)

[4] Birger Andersson, Maria Bergholtz, Ananda Edirisuriya, Tharaka Ilayperuma, and Paul Johannesson. A Declarative Foundation of Process Models. In *Advanced Information Systems Engineering*, volume 3520 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2005. (Cited on page 78.)

[5] Richard C. Angell, George E. Freund, and Peter Willett. Automatic Spelling Correction Using a Trigram Similarity Measure. *Information Processing & Management*, 19(4):255 – 261, 1983. (Cited on page 58.)

[6] Ahmed Awad. BPMN-Q: A Language to Query Business Processes. In *EMISA*, volume 119, pages 115–128, 2007. (Cited on pages 35, 106, 108, 114, 207, and 235.)

[7] Ahmed Awad. *A Compliance Management Framework for Business Process Models*. PhD thesis, University of Potsdam, 2010. (Cited on pages 26, 32, and 115.)

[8] Ahmed Awad and Sherif Sakr. Querying Graph-Based Repositories of Business Process Models. In *Database Systems for Advanced Applications*, volume 6193 of *Lecture Notes in Computer Science*, pages 33–44. Springer, 2010. (Cited on pages 109, 114, 136, and 235.)

[9] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In *Business Process Management*, volume 5240 of *Lecture Notes in Com-*

puter Science, pages 326–341. Springer, 2008. (Cited on pages 109 and 117.)

[10] Ahmed Awad, Artem Polyvyanyy, and Mathias Weske. Semantic Querying of Business Process Models. In *Enterprise Distributed Object Computing*, pages 85–94. IEEE Computer Society, 2008. (Cited on pages 114, 136, 139, and 235.)

[11] Ahmed Awad, Gero Decker, and Niels Lohmann. Diagnosing and Repairing Data Anomalies in Process Models. In *Business Process Management Workshops*, volume 43 of *Lecture Notes in Business Information Processing*, pages 5–16. Springer, 2009. (Cited on page 119.)

[12] Ahmed Awad, Sherif Sakr, Matthias Kunze, and Mathias Weske. Design by Selection: A Reuse-Based Approach for Business Process Modeling. In *Conceptual Modeling (ER)*, volume 6998 of *Lecture Notes in Computer Science*, pages 332–345. Springer, 2011. (Cited on pages 9, 61, 88, 106, 107, and 109.)

[13] Joonsoo Bae, James Caverlee, Ling Liu, and Hua Yan. Process Mining by Measuring Process Block Similarity. In *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 141–152. Springer, 2006. (Cited on pages 147 and 238.)

[14] Joonsoo Bae, Ling Liu, James Caverlee, and William B. Rouse. Process Mining, Discovery, and Integration using Distance Measures. In *Web Services*, pages 479–488. IEEE Computer Society, 2006. (Cited on pages 148 and 238.)

[15] Joonsoo Bae, Ling Liu, James Caverlee, Liang-Jie Zhang, and Hyerim Bae. Development of Distance Measures for Process Mining, Discovery and Integration. *Int. J. Web Service Res.*, 4(4): 1–17, 2007. (Cited on pages 136, 144, 148, and 238.)

[16] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval - The concepts and Technology Behind Search*. Pearson Education Ltd., 2nd edition edition, 2011. (Cited on pages 62, 138, 139, 142, 146, 178, and 182.)

[17] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008. (Cited on pages 26, 65, 115, and 121.)

[18] Wasana Bandara, Guy G. Gable, and Michael Rosemann. Critical Success Factors of Business Process Modeling. Technical report, Queensland University of Technology, 2007. (Cited on page 29.)

[19] Twan Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. PhD thesis, Eindhoven University of Technology, 1998. (Cited on pages 116, 118, and 121.)

[20] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Comput. Surv.*, 18(4):323–364, December 1986. (Cited on page 54.)

[21] Jörg Becker, Philipp Bergener, Dominic Breuker, and Michael Räckers. On Measures of Behavioral Distance between Business Processes. In *Wirtschaftsinformatik*, pages 665–674, 2011. (Cited on pages 34, 143, 150, 153, and 239.)

[22] Michael Becker and Ralf Laue. A Comparative Survey of Business Process Similarity Measures. *Computers in Industry*, 63(2): 148–167, 2012. (Cited on pages 34, 37, 68, 93, 119, and 146.)

[23] Catriel Beeri, Anat Eyal, Simon Kamenkovich, and Tova Milo. Querying Business Processes. In *Very Large Data Bases*, pages 343–354. ACM, 2006. (Cited on pages 114 and 235.)

[24] Catriel Beeri, Anat Eyal, Tova Milo, and Alon Pilberg. Monitoring Business Processes with Queries. In *Very Large Data Bases*, pages 603–614. ACM, 2007. (Cited on page 109.)

[25] Catriel Beeri, Anat Eyal, Simon Kamenkovich, and Tova Milo. Querying Business Processes with BP-QL. *Inf. Syst.*, 33(6):477–507, September 2008. (Cited on pages 39, 40, 96, 114, 126, and 235.)

[26] Khalid Belhajjame and Marco Brambilla. Ontology-Based Description and Discovery of Business Processes. In *Enterprise, Business-Process and Information Systems Modeling*, volume 29 of *Lecture Notes in Business Information Processing*, pages 85–98. Springer, 2009. (Cited on pages 40, 57, 113, and 234.)

[27] Jochen Van den Bercken, Björn Blohsfeld, Jens-Peter Dittrich, Jürgen Krämer, Tobias Schäfer, Martin Schneider, and Bernhard Seeger. XXL - A Library Approach to Supporting Efficient Implementations of Advanced Database Queries. In *Very Large Data Bases*, pages 39–48. Morgan Kaufmann, 2001. (Cited on pages 192 and 197.)

[28] Ralph Bergmann and Yolanda Gil. Retrieval of Semantic Workflows with Knowledge Intensive Similarity Measures. In *Case-Based Reasoning Research and Development*, volume 6880 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2011. (Cited on pages 147 and 237.)

[29] Lasse Bergroth, Harri Hakonen, and Timo Raita. A Survey of Longest Common Subsequence Algorithms. In *String Processing Information Retrieval*, SPIRE '00, pages 39–48. IEEE Computer Society, 2000. (Cited on page 64.)

[30] Abraham Bernstein, Esther Kaufmann, Christoph Bürki, and Mark Klein. How Similar Is It? Towards Personalized Similarity Measures in Ontologies. In *Wirtschaftsinformatik*, pages 1347–1366. Physica-Verlag, 2005. (Cited on page 142.)

[31] Philip A. Bernstein and Umeshwar Dayal. An overview of repository technology. In *Very Large Data Bases*, VLDB '94, pages 705–713. Morgan Kaufmann Publishers Inc., 1994. (Cited on pages 4 and 30.)

[32] Philip A. Bernstein and Erhard Rahm. Data Warehouse Scenarios for Model Management. In *Conceptual Modeling (ER)*, volume 1920 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2000. (Cited on page 54.)

[33] Eike Best. Structure theory of Petri nets: the free choice hiatus. In *Petri Nets: Central Models and Their Properties*, volume 254 of *Lecture Notes in Computer Science*, pages 168–205. Springer, 1987. (Cited on page 49.)

[34] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When Is "Nearest Neighbor" Meaningful? In *In Int. Conf. on Database Theory*, pages 217–235, 1999. (Cited on pages 88, 91, and 92.)

[35] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Trace Alignment in Process Mining: Opportunities for Process Diagnostics. In *Business Process Management*, volume 6336 of *Lecture Notes in Computer Science*, pages 227–242. Springer, 2010. (Cited on pages 122 and 130.)

[36] Alessandro Bozzon, Marco Brambilla, and Piero Fraternali. Searching Repositories of Web Application Models. In *Web Engineering*, volume 6189 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2010. (Cited on page 36.)

[37] Sergey Brin. Near Neighbor Search in Large Metric Spaces. In *Very Large Data Bases*, pages 574–584. Morgan Kaufmann, 1995. (Cited on page 168.)

[38] Saartje Brockmans, Marc Ehrig, Agnes Koschmider, Andreas Oberweis, and Rudi Studer. Semantic Alignment Of Business Processes. In *Enterprise Information Systems*, pages 191–196. INSTICC Press, 2006. (Cited on page 57.)

[39] Chris Buckley and Ellen M. Voorhees. Evaluating Evaluation Measure Stability. In *Information Retrieval*, pages 33–40. ACM, 2000. (Cited on pages 178 and 180.)

[40] Horst Bunke and G. Allermann. Inexact Graph Matching for Structural Pattern Recognition. *Pattern Recognition Letters*, 1(4): 245 – 253, 1983. (Cited on pages 34, 64, 113, 146, 148, 149, and 200.)

[41] Horst Bunke and Bruno T. Messmer. Similarity Measures for Structured Representations. In *Topics in Case-Based Reasoning*, volume 837 of *Lecture Notes in Computer Science*, pages 106–118. Springer, 1994. (Cited on pages 113, 118, and 146.)

[42] Horst Bunke and Kim Shearer. A Graph Distance Metric Based on the Maximal Common Subgraph. *Pattern Recogn. Lett.*, 19: 255–259, March 1998. (Cited on pages 146 and 148.)

[43] Rainer E. Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. SIAM e-books. Society for Industrial and Applied Mathematics, revised reprint. edition, 2012. (Cited on page 58.)

[44] Walter A. Burkhard and Robert M. Keller. Some Approaches to Best-Match File Searching. *Commun. ACM*, 16:230–236, April 1973. (Cited on page 168.)

[45] Donald D. Chamberlin and Raymond F. Boyce. SEQUEL: A Structured English Query Language. In *Workshop on Data description, Access and Control*, pages 249–264. ACM, 1974. (Cited on page 112.)

[46] Gary Chartrand, Grzegorz Kubicki, and Michelle Schultz. Graph Similarity and Distance in Graphs. *Aequationes Mathematicae*, 55:129–145, 1998. (Cited on page 146.)

[47] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in Metric Spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001. (Cited on pages 142, 143, 167, and 197.)

[48] David Chiu, Travis Hall, Farhana Kabir, and Gagan Agrawal. An Approach towards Automatic Workflow Composition through Information Retrieval. In *Database Engineering & Applications*, pages 170–178. ACM, 2011. (Cited on pages 112 and 234.)

[49] Injun Choi, Kwangmyeong Kim, and Mookyung Jang. An XML-Based Process Repository and Process Query Language for Integrated Process Management. *Knowledge and Process Manage-*

ment, 14(4):303–316, 2007. (Cited on pages 4, 108, 109, 115, 126, and 235.)

[50] Namyoun Choi, Il-Yeol Song, and Hyoil Han. A Survey on Ontology Mapping. *SIGMOD Rec.*, 35(3):34–41, September 2006. (Cited on pages 54 and 82.)

[51] Vassilis Christophides, Richard Hull, and Akhil Kumar. Querying and Splicing of XML Workflows. In *Cooperative Information Systems*, volume 2172 of *Lecture Notes in Computer Science*, pages 386–403. Springer, 2001. (Cited on pages 109, 114, and 235.)

[52] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Very Large Data Bases*, pages 426–435. Morgan Kaufmann, 1997. (Cited on pages 168, 169, 183, and 197.)

[53] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. Mit Press, 1999. (Cited on page 115.)

[54] Edgar F. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, June 1970. (Cited on page 112.)

[55] William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Workshop on Information Integration on the Web*, pages 73–78, 2003. (Cited on page 58.)

[56] David Cohn and Richard Hull. Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. *IEEE Data Engineering Bulletin*, 32(3):3–9, 2009. (Cited on pages 24 and 71.)

[57] William J. Conover. *Practical Non-Parametric Statistics*. John Wiley and Sons, Second edition, 1980. (Cited on page 95.)

[58] Flavio Costa, Daniel Oliveira, Eduardo Ogasawara, AlexandreA.B. Lima, and Marta Mattoso. Athena: Text Mining Based Discovery of Scientific Workflows in Disperse Repositories. In *Resource Discovery*, volume 6799 of *Lecture Notes in Computer Science*, pages 104–121. Springer, 2012. (Cited on page 112.)

[59] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley, 2010. (Cited on page 88.)

[60] Thomas Curran, Gerhard Keller, and Andrew Ladd. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Prentice-Hall, Inc., 1997. (Cited on pages 184, 186, 187, and 200.)

[61] Alfredo Cuzzocrea, Juri Luca De Coi, Marco Fisichella, and Dimitrios Skoutas. Graph-Based Matching of Composite OWL-S Services. In *Database Systems for Advanced Applications*, volume 6637 of *Lecture Notes in Computer Science*, pages 28–39. Springer, 2011. (Cited on pages 57, 148, and 238.)

[62] Ajantha Dahanayake, Richard J. Welke, and Gabriel Cavalheiro. Improving the Understanding of BAM Technology for Real-Time Decision Support. *Int. J. Bus. Inf. Syst.*, 7:1–26, December 2011. (Cited on page 28.)

[63] Thomas H. Davenport. *Process Innovation: Reengineering Work through Information Technology*. Harvard Business School Publishing India Pvt. Limited, 1993. (Cited on pages 4, 16, 17, 18, 20, 23, and 27.)

[64] Islay Davies, Peter Green, Michael Rosemann, Marta Indulska, and Stan Gallo. How do practitioners use conceptual modeling in practice? *Data Knowl. Eng.*, 58(3):358–380, September 2006. (Cited on page 71.)

[65] Jos de Bruijn. Web Service Modeling Language (WSML), version 1.0. `http://www.wsmo.org/TR/d16/d16.3/v1.0/`, August 2008. (Cited on page 115.)

[66] Ana. K. A. de Medeiros, W. M. P. van der Aalst, and A. J. M. M. Weijters. Quantifying Process Equivalence Based on Observed Behavior. *Data Knowl. Eng.*, 64(1):55–74, 2008. (Cited on pages 34, 62, 63, 64, 83, 96, 118, 146, 147, 150, 153, 237, and 239.)

[67] Gero Decker and Jan Mendling. Process Instantiation. *Data Knowl. Eng.*, 68:777–792, September 2009. (Cited on pages 72, 77, and 185.)

[68] Jörg Desel and Javier Esparza. *Free Choice Petri Nets*. Cambridge University Press, 1995. (Cited on pages 45, 48, 49, and 73.)

[69] Daniel Deutch and Tova Milo. Evaluating TOP-K Queries over Business Processes. In *Data Engineering*, pages 1195–1198. IEEE Computer Society, 2009. (Cited on pages 117 and 236.)

[70] Daniel Deutch and Tova Milo. TOP-K projection queries for probabilistic business processes. In *Database Theory*, pages 239–251. ACM, 2009. (Cited on pages 109, 117, 118, and 236.)

[71] Daniel Deutch and Tova Milo. A Structural/Temporal Query Language for Business Processes. *J. Comput. Syst. Sci.*, 78(2): 583–609, March 2012. (Cited on pages 114 and 235.)

[72] Remco Dijkman. A Classification of Differences between Similar Business Processes. In *Enterprise Distributed Object Computing Conference*, pages 37–50. IEEE Computer Society, 2007. (Cited on pages 55, 74, 76, 77, 78, and 146.)

[73] Remco Dijkman, Marlon Dumas, and Luciano García-Bañuelos. Graph Matching Algorithms for Business Process Model Similarity Search. In *Business Process Management*, volume 5701 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 2009. (Cited on pages 58, 96, 142, 149, 200, and 238.)

[74] Remco Dijkman, Marlon Dumas, Luciano Garcia-Banuelos, and Reina Kaarik. Aligning Business Process Models. In *Enterprise Distributed Object Computing Conference*, pages 45–53. IEEE Computer Society, 2009. (Cited on pages 55, 149, and 238.)

[75] Remco Dijkman, Marlon Dumas, Boudewijn F. van Dongen, Reina Käärik, and Jan Mendling. Similarity of Business Process Models: Metrics and Evaluation. *Information Systems*, 36 (2):498 – 516, 2011. Special Issue: Semantic Integration of Data, Multimedia, and Services. (Cited on pages 34, 57, 58, 63, 149, 151, 153, 183, 184, 191, 199, 200, 204, 238, and 240.)

[76] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Inf. Softw. Technol.*, 50(12):1281–1294, 2008. (Cited on pages 72, 119, 128, and 142.)

[77] Remco M. Dijkman, Marcello La Rosa, and Hajo A. Reijers. Managing Large Collections of Business Process Models - Current Techniques and Challenges. *Computers in Industry*, 63(2): 91–97, 2012. (Cited on pages 4, 29, 30, 32, and 35.)

[78] Remco M. Dijkman, Boudewijn F. van Dongen, Luciano Garcia-Banuelos Dumas, Marlon and, Matthias Kunze, Henrik Leopold, Jan Mendling, Reina Uba, Matthias Weidlich, Mathias Weske, and Zhiqiang Yan. A Short Survey on Process Model Similarity. In *Seminal Contributions to Information Systems Engineering*, pages 421–427. Springer, 2013. (Cited on pages 9, 141, and 151.)

[79] Marlon Dumas, Luciano García-Bañuelos, and Remco M. Dijkman. Similarity Search of Business Process Models. *IEEE Data Eng. Bull.*, 32(3):23–28, 2009. (Cited on pages 33, 37, 55, 57, 143, and 146.)

[80] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013. (Cited on pages 16, 20, 22, 27, and 29.)

[81] Marc Ehrig, Agnes Koschmider, and Andreas Oberweis. Measuring Similarity between Semantic Business Process Models. In *Asia-Pacific Conference on Conceptual Modelling*, volume 67 of *CRPIT*, pages 71–80. Australian Computer Society, Inc., 2007. (Cited on pages 33, 34, 57, 58, 147, and 237.)

[82] Chathura C. Ekanayake, Marlon Dumas, Luciano García-Bañuelos, Marcello Rosa, and ArthurH.M. Hofstede. Approximate Clone Detection in Repositories of Business Process Models. In *Business Process Management*, volume 7481 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 2012. (Cited on pages 34, 113, 117, 143, 149, and 234.)

[83] Eren Esgin and Pinar Senkul. Delta Analysis: A Hybrid Quantitative Approach for Measuring Discrepancies between Business Process Models. In *Hybrid Artificial Intelligent Systems*, volume 6678 of *Lecture Notes in Computer Science*, pages 296–304. Springer, 2011. (Cited on pages 32, 147, and 238.)

[84] Rik Eshuis and Paul Grefen. Structural Matching of BPEL Processes. In *European Conference on Web Services*, pages 171–180. IEEE Computer Society, 2007. (Cited on pages 8, 40, 41, 52, 67, 143, 152, 153, 154, and 240.)

[85] Javier Esparza, Stefan Römer, and Walter Vogler. An Improvement of McMillan's Unfolding Algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002. (Cited on pages 67, 116, and 117.)

[86] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer, 2007. (Cited on pages 39, 54, 55, 57, 58, 62, 65, 85, 146, 157, and 178.)

[87] Dirk Fahland, Cédric Favre, Barbara Jobstmann, Jana Koehler, Niels Lohmann, Hagen Völzer, and Karsten Wolf. Instantaneous Soundness Checking of Industrial Business Process Models. In *Business Process Management*, volume 5701 of *Lecture Notes in Computer Science*, pages 278–293. Springer, 2009. (Cited on page 26.)

[88] Peter Fettke, Peter Loos, and Jörg Zwicker. Business Process Reference Models: Survey and Classification. In *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 469–483. Springer, 2005. (Cited on pages 32 and 144.)

[89] Organization for the Advancement of Structured Information Standards (OASIS). OASIS Web Services Business Process Execution Language. http://docs.oasis-

open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, April 2007. (Cited on pages 25, 71, and 143.)

[90] Jacob Freund and Bernd Rücker. *Real-Life BPMN: Using BPMN 2.0 to Analyze, Improve, and Automate Processes in Your Company*. CreateSpace Independent Publishing Platform, 2012. (Cited on pages 16 and 20.)

[91] Timo Füermann and Carsten Dammasch. *Prozessmanagement: Anleitung zur ständigen Prozessverbesserung*. Number 9783446415713 in Pocket Power. Hanser Fachbuchverlag, 2008. (Cited on pages 16, 18, 19, 20, and 27.)

[92] George W. Furnas, Thomas K. Landauer, Louis M. Gomez, and Susan Dumais. The Vocabulary Problem in Human-System Communication. *Commun. ACM*, 30(11):964–971, November 1987. (Cited on pages 34, 37, 39, 57, 138, and 157.)

[93] Avigdor Gal and Tomer Sagi. Tuning the Ensemble Selection Process of Schema Matchers. *Inf. Syst.*, 35(8):845–859, December 2010. (Cited on pages 55 and 57.)

[94] Juntao Gao and Li Zhang. On Measuring Semantic Similarity of Business Process Models. In *Interoperability for Enterprise Software and Applications*, pages 289–293. IEEE Computer Society, 2009. (Cited on pages 33, 147, and 237.)

[95] Juntao Gao, Li Zhang, and Wei Jiang. Procuring Requirements for ERP Software Based on Semantic Similarity. In *Semantic Computing*, pages 61–70. IEEE Computer Society, 2007. (Cited on pages 147 and 237.)

[96] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A Survey of Graph Edit Distances. *Pattern Analysis & Applications*, 13(1): 113–129, February 2010. (Cited on pages 80, 146, and 148.)

[97] Kerstin Gerke, Jorge Cardoso, and Alexander Claus. Measuring the Compliance of Processes with Reference Models. In *On the Move to Meaningful Internet Systems*, volume 5870 of *Lecture Notes in Computer Science*, pages 76–93. Springer, 2009. (Cited on pages 40, 64, 73, 144, 150, 153, and 239.)

[98] Yolanda Gil, Jihie Kim, Gonzalo Florez, Varun Ratnakar, and Pedro A. González-Calero. Workflow Matching Using Semantic Metadata. In *Knowledge Capture*, pages 121–128. ACM, 2009. (Cited on page 109.)

[99] Antoon Goderis. *Workflow Re-use and Discovery in Bioinformatics*. PhD thesis, School of Computer Science, The University of Manchester, 2008. (Cited on page 41.)

[100] Antoon Goderis, Ulrike Sattler, Phillip Lord, and Carole Goble. Seven Bottlenecks to Workflow Reuse and Repurposing. In *The Semantic Web – ISWC*, volume 3729 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2005. (Cited on pages 31 and 40.)

[101] Antoon Goderis, Peter Li, and Carole Goble. Workflow Discovery: The Pproblem, a Case Study from e-Science and a Graph-Based Solution. In *Web Services*, pages 312–319. IEEE Computer Society, 2006. (Cited on page 109.)

[102] Guido Governatori, Zoran Milosevic, and Shazia Sadiq. Compliance Checking Between Business Processes and Business Contracts. In *Enterprise Distributed Object Computing Conference*, pages 221–232. IEEE Computer Society, 2006. (Cited on pages 26 and 32.)

[103] Daniela Grigori, Fabio Casati, Malu Castellanos, Umeshwar Dayal, Mehmet Sayal, and Ming-Chien Shan. Business Process Intelligence. *Computers in Industry*, 53(3):321–343, April 2004. (Cited on page 28.)

[104] Daniela Grigori, Juan Carlos Corrales, and Mokrane Bouzeghoub. Behavioral Matchmaking for Service Retrieval. In *Web Services*, pages 145–152. IEEE Computer Society, 2006. (Cited on pages 113, 118, and 235.)

[105] Daniela Grigori, Juan Carlos Corrales, Mokrane Bouzeghoub, and Ahmed Gater. Ranking BPEL Processes for Service Discovery. *IEEE Transactions on Services Computing*, 3:178–192, 2010. (Cited on pages 34, 113, 118, 143, 157, 183, and 235.)

[106] Object Management Group. Unified Modeling Language (UML), v2.4.1. http://www.omg.org/spec/UML/2.4.1/, August 2011. (Cited on pages 24, 30, and 71.)

[107] Varun Grover, Kirk D. Fiedler, and James T. C. Teng. Exploring the Success of Information Technology Enabled Business Process Reengineering. *IEEE Transactions on Engineering Management*, 41(3):276–284, 1994. (Cited on page 16.)

[108] Volker Gruhn and Monika Schneider. Workflow Management Based on Process Model Repositories. In *Software Engineering*, pages 379–388. IEEE Computer Society, 1998. (Cited on page 29.)

[109] Thomas Gschwind, Jana Koehler, and Janette Wong. Applying Patterns during Business Process Modeling. In *Business Process Management*, Lecture Notes in Computer Science, pages 4–19. Springer, 2008. (Cited on page 34.)

[110] Markus Guentert, Matthias Kunze, and Mathias Weske. Evaluation Measures for Similarity Search Results in Process Model Repositories. In *Conceptual Modeling (ER)*, volume 7532 of *Lecture Notes in Computer Science*, pages 214–227. Springer, 2012. (Cited on pages 9, 61, 177, and 182.)

[111] Michael H. T. Hack. Analysis of Production Schemata by Petri Nets. Technical report, Massachusetts Institute of Technology, 1972. (Cited on page 49.)

[112] Michel H. T. Hack. *Decidability Questions for Petri Nets*. PhD thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science, 1976. (Cited on pages 45, 47, 49, 53, 101, and 123.)

[113] Michael Hammer. What is Business Process Management? In *Handbook on Business Process Management 1*, International Handbooks on Information Systems, pages 3–16. Springer, 2010. (Cited on pages 16, 17, and 20.)

[114] Michael Hammer and James Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. HarperBusiness Essential. HarperBusiness, rev. and updated with a new authors' note edition, 1993. (Cited on pages 4, 16, 17, 18, 19, 20, 23, 27, 28, and 31.)

[115] Yanan Hao and Yanchun Zhang. Web Services Discovery Based on Schema Matching. In *Australasian Conference on Computer Science*, pages 107–113. Australian Computer Society, Inc., 2007. (Cited on page 54.)

[116] David Harel and Bernhard Rumpe. Meaningful Modeling: What's the Semantics of "Semantics"? *Computer*, 37(10):64–72, October 2004. (Cited on pages 24 and 33.)

[117] Scott Henninger. An Evolutionary Approach to Constructing Effective Software Reuse Repositories. *ACM Trans. Softw. Eng. Methodol.*, 6(2):111–140, 1997. (Cited on pages 33, 34, and 37.)

[118] Martin Hepp, Frank Leymann, John Domingue, Alexander Wahler, and Dieter Fensel. Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In *e-Business Engineering*, pages 535–540. IEEE Computer Society, 2005. (Cited on page 36.)

[119] Nico Herzberg, Matthias Kunze, and Andreas Rogge-Solti. Towards Process Evaluation in Non-automated Process Execution Environments. In *Central-European Workshop on Services and their Composition*, volume 847, pages 96–102. ceur-ws.org, 2012. (Cited on pages 18 and 28.)

[120] Jan Hidders, Marlon Dumas, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Jan Verelst. When are two workflows the same? In *Australasian Symposium on Theory of Computing*, pages 3–11. Australian Computer Society, Inc., 2005. (Cited on pages 63, 64, and 65.)

[121] Sebastian Hinz, Karsten Schmidt, and Christian Stahl. Transforming BPEL to Petri Nets. In *Business Process Management*, volume 3649 of *Lecture Notes in Computer Science*, pages 220–235. Springer, 2005. (Cited on page 72.)

[122] Gisli R. Hjaltason and Hanan Samet. Index-Driven Similarity Search in Metric Spaces. *ACM Trans. Database Syst.*, 28(4):517–580, 2003. (Cited on pages 143, 167, and 197.)

[123] Charles A. R. Hoare. Communicating Sequential Processes. *Commun. ACM*, 21(8):666–677, August 1978. (Cited on page 63.)

[124] Charles A. R. Hoare. Some Properties of Predicate Transformers. *J. ACM*, 25(3):461–480, July 1978. (Cited on pages 64 and 146.)

[125] Charles A. R. Hoare. A Model for Communicating Sequential Processes. Technical report, Oxford University Computing Laboratory, 1980. (Cited on pages 47, 63, 64, 76, 116, 119, 130, 146, and 149.)

[126] Thomas Hornung, Agnes Koschmider, and Georg Lausen. Recommendation Based Process Modeling Support: Method and User Experience. In *Conceptual Modeling (ER)*, volume 5231 of *Lecture Notes in Computer Science*, pages 265–278. Springer, 2008. (Cited on pages 147 and 237.)

[127] David. A. Huffman. The Synthesis of Sequential Switching Circuits. Technical Report 274, Massachusetts Institute of Technology, 1954. (Cited on page 63.)

[128] Bernhard G. Humm and Janina Fengel. Semantics-Based Business Process Model Similarity. In *Business Information Systems*, volume 117 of *Lecture Notes in Business Information Processing*, pages 36–47. Springer, 2012. (Cited on pages 33, 143, 147, and 237.)

[129] M. J. Ibáñez, G. Vulcu, J. Ezpeleta, and S. Bhiri. Semantically Enabled Business Process Discovery. In *ACM Symposium on Applied Computing*, pages 1396–1403. ACM, 2010. (Cited on pages 33, 109, 115, and 236.)

[130] Marta Indulska, Peter Green, Jan Recker, and Michael Rosemann. Business Process Modeling: Perceived Benefits. In *Conceptual Modeling (ER)*, volume 5829 of *Lecture Notes in Computer*

Science, pages 458–471. Springer, 2009. (Cited on pages 26, 27, and 28.)

[131] Marta Indulska, Jan Recker, Michael Rosemann, and Peter Green. Business process modeling: Current issues and future challenges. In *Advanced Information Systems Engineering*, volume 5565 of *Lecture Notes in Computer Science*, pages 501–514. Springer, 2009. (Cited on pages 6, 23, 24, 28, and 30.)

[132] P. Jaccard. Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:241–272, 1901. (Cited on pages 67, 151, and 155.)

[133] Tao Jin, Jianmin Wang, Nianhua Wu, Marcello La Rosa, and Arthur H. M. ter Hofstede. Efficient and Accurate Retrieval of Business Process Models through Indexing. In *On the Move to Meaningful Internet Systems*, volume 6426 of *Lecture Notes in Computer Science*, pages 402–409. Springer, 2010. (Cited on pages 109, 113, 117, 136, 183, 184, and 234.)

[134] Tao Jin, Jianmin Wang, and Lijie Wen. Efficient Retrieval of Similar Business Process Models Based on Structure. In *On the Move to Meaningful Internet Systems*, volume 7044 of *Lecture Notes in Computer Science*, pages 56–63. Springer Berlin, 2011. (Cited on pages 136, 148, 183, and 238.)

[135] Tao Jin, Jianmin Wang, and Lijie Wen. Querying Business Process Models Based on Semantics. In *Database Systems for Advanced Applications*, volume 6588 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 2011. (Cited on pages 33, 53, 67, 68, 116, 117, 139, and 236.)

[136] Tao Jin, Jianmin Wang, and Lijie Wen. Efficient Retrieval of Similar Workflow Models Based on Behavior. In *Web Technologies and Applications*, volume 7235 of *Lecture Notes in Computer Science*, pages 677–684. Springer, 2012. (Cited on pages 67, 73, 136, 144, and 183.)

[137] Jae-Yoon Jung and Joonsoo Bae. Workflow Clustering Method Based on Process Similarity. In *Computational Science and Its Applications*, volume 3981 of *Lecture Notes in Computer Science*, pages 379–389. Springer, 2006. (Cited on pages 147 and 238.)

[138] Jae-Yoon Jung, Joonsoo Bae, and Ling Liu. Hierarchical Business Process Clustering. In *Services Computing*, volume 2, pages 613–616. IEEE Computer Society, july 2008. (Cited on pages 147 and 238.)

[139] Jae-Yoon Jung, Joonsoo Bae, and Ling Liu. Hierarchical Clustering of Business Process Models. *International Journal of Innovative Computing, Information and Control*, 5(12):4501–4511, December 2009. (Cited on pages 143, 144, 147, and 238.)

[140] Iraj Kalantari and Gerard McDonald. A Data Structure and an Algorithm for the Nearest Point Problem. *IEEE Transactions on Software Engineering*, 9:631–634, 1983. (Cited on page 168.)

[141] Kathrin Kaschner and Karsten Wolf. Set Algebra for Service Behavior: Applications and Constructions. In *Business Process Management*, volume 5701 of *Lecture Notes in Computer Science*, pages 193–210. Springer, 2009. (Cited on page 34.)

[142] G. Keller, M. Nüttgens, and A.-W. Scheer. Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Technical Report 89, Institut für Wirtschaftsinformatik, 1992. (Cited on pages 24 and 71.)

[143] John L. Kelley. *General Topology*. University series in higher mathematics. Van Nostrand, 1955. (Cited on page 166.)

[144] Zaheer Khan, Mohammed Odeh, and Richard McClatchey. Bridging the Gap Between Business Process Models and Service-Oriented Architectures with Reference to the Grid Environment. *Int. J. Grid Util. Comput.*, 2(4):253–283, October 2011. (Cited on page 72.)

[145] Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Wil M. P. van der Aalst. Fundamentals of Control Flow in Workflows. *Acta Informatica*, 39:143–209, 2002. (Cited on pages 48, 50, and 102.)

[146] Mark Klein and Abraham Bernstein. Toward High-Precision Service Retrieval. *IEEE Internet Computing*, 8:30–36, 2004. (Cited on pages 39, 112, and 234.)

[147] Andreas Knöpfel, Bernhard Gröne, and Peter Tabeling. *Fundamental Modeling Concepts. Effective Communication of IT Systems*. John Wiley, 2005. (Cited on pages 29, 206, and 219.)

[148] Donald E. Knuth. *The Art of Computer Programming. Vol.3: Sorting and Searching*, volume 3. Addison-Wesley, 2nd edition edition, 1973. (Cited on pages 32, 33, 112, and 136.)

[149] Agnes Koschmider. *Ähnlichkeitsbasierte Modellierungsunterstützung für Geschäftsprozesse*. PhD thesis, Universität Karlsruhe (TH), Fakultät für Wirtschaftswissenschaften, 2007. (Cited on pages 31 and 109.)

[150] Agnes Koschmider, Thomas Hornung, and Andreas Oberweis. Recommendation-Based Editor for Business Process Modeling. *Data Knowl. Eng.*, 70(6):483–503, June 2011. (Cited on pages 33, 112, and 234.)

[151] Eugene F. Krause. *Taxicab Geometry*. Addison-Wesley Innovative Series. Addison-Wesley Pub. Co., 1975. (Cited on page 148.)

[152] Matthias Kunze and Mathias Weske. Entwurf domänenspezifischer Modelle im Web mit Oryx. In *GI Jahrestagung*, volume 154 of *Lecture Notes in Informatics*, pages 2960–2971. Gesellschaft für Informatik, 2009. (Cited on page 24.)

[153] Matthias Kunze and Mathias Weske. Metric Trees for Efficient Similarity Search in Process Model Repositories. In *Business Process Management Workshops*, volume 66 of *Lecture Notes in Business Information Processing*, pages 535–546. Springer, September 2010. (Cited on pages 9, 141, 149, 152, 154, 177, 184, and 197.)

[154] Matthias Kunze and Mathias Weske. Local Behavior Similarity. In *Enterprise, Business-Process and Information Systems Modeling*, volume 113 of *Lecture Notes in Business Information Processing*, pages 107–120. Springer, 2012. (Cited on pages 9, 35, 61, 68, 107, 112, and 118.)

[155] Matthias Kunze and Mathias Weske. Methods for Evaluating Process Model Search. In *Business Process Management Workshops*, Lecture Notes in Business Information Processing. Springer, 2013. Accepted for publication, contact the authors for more information. (Cited on pages 9, 117, 177, and 178.)

[156] Matthias Kunze and Mathias Weske. Visualization of Successor Relations in Business Process Models. In *Web Services and Formal Methods*, Lecture Notes in Computer Science, page (to appear). Springer, 2013. Accepted for publication, contact the authors for more information. (Cited on pages 9, 40, and 61.)

[157] Matthias Kunze, Alexander Luebbe, Matthias Weidlich, and Mathias Weske. Towards Understanding Process Modeling – The Case of the BPM Academic Initiative. In *Business Process Model and Notation*, volume 95 of *Lecture Notes in Business Information Processing*, pages 44–58. Springer, 2011. (Cited on pages 9, 77, and 177.)

[158] Matthias Kunze, Matthias Weidlich, and Mathias Weske. Behavioral Similarity – A Proper Metric. In *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2011. (Cited on pages 9, 52, 61, 68, 141, 142, 152, 153, 154, 155, 177, 184, 191, 204, and 240.)

[159] Matthias Kunze, Matthias Weidlich, and Mathias Weske. m3 - A Behavioral Similarity Metric for Business Processes. In *Central-European Workshop on Services and their Composition*, volume 705, pages 89–95. ceur-ws.org, Feb 2011. (Cited on pages 9, 68, 141, 152, 153, and 240.)

[160] Matthias Kunze, Matthias Weidich, and Mathias Weske. Querying Process Models by Behavior Inclusion. *Software and System Modeling (SoSym)*, 2013. Accepted for publication, contact the authors for more information. (Cited on pages 9, 107, and 177.)

[161] Jochen M. Küster, Christian Gerth, Alexander Förster, and Gregor Engels. Detecting and Resolving Process Model Differences in the Absence of a Change Log. In *Business Process Management*, volume 5240 of *Lecture Notes in Computer Science*, pages 244–260. Springer, 2008. (Cited on pages 31, 74, 76, 77, 149, and 239.)

[162] Marcello La Rosa. *Managing Variability in Process-Aware Information Systems*. PhD thesis, Queensland University of Technology, 2009. (Cited on pages 32 and 144.)

[163] Marcello La Rosa, Hajo A. Reijers, Wil M. P. van der Aalst, Remco M. Dijkman, Jan Mendling, Marlon Dumas, and Luciano García-Bañuelos. APROMORE : An Advanced Process Model Repository. *Expert Syst. Appl.*, 38(6):7029–7040, June 2011. (Cited on pages 4, 22, 30, 39, and 142.)

[164] Jill H. Larkin and Herbert A. Simon. Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science*, 11(1):65–100, 1987. (Cited on page 24.)

[165] Ralf Laue and Ahmed Awad. Visualization of Business Process Modeling Anti Patterns. *ECEASST*, 25, 2010. (Cited on page 96.)

[166] Henrik Leopold, Jan Mendling, and Hajo A. Reijers. On the Automatic Labeling of Process Models. In *Advanced Information Systems Engineering*, volume 6741 of *Lecture Notes in Computer Science*, pages 512–520. Springer, 2011. (Cited on page 184.)

[167] Henrik Leopold, Mathias Niepert, Matthias Weidlich, Jan Mendling, Remco M. Dijkman, and Heiner Stuckenschmidt. Probabilistic Optimization of Semantic Process Model Matching. In *Business Process Management*, volume 7481 of *Lecture Notes in Computer Science*, pages 319–334. Springer, 2012. (Cited on pages 34 and 58.)

[168] Vladimir I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10: 707, 1966. (Cited on pages 58, 149, 151, 159, and 200.)

[169] Chen Li, Manfred Reichert, and Andreas Wombacher. On Measuring Process Model Similarity Based on High-Level Change Operations. In *Conceptual Modeling (ER)*, volume 5231 of *Lecture Notes in Computer Science*, pages 248–264. Springer, 2008. (Cited on pages 96, 149, and 239.)

[170] Xitong Li, Yushun Fan, Quan Z. Sheng, Zakaria Maamar, and Hongwei Zhu. A Petri Net Approach to Analyzing Behavioral Compatibility and Similarity of Web Services. *IEEE Transactions on Systems, Man, and Cybernetics*, 41(3):510–521, 2011. (Cited on pages 34, 64, 143, 149, 150, 153, and 239.)

[171] Dekang Lin. An Information-Theoretic Definition of Similarity. In *Machine Learning*, pages 296–304. Morgan Kaufmann Publishers Inc., 1998. (Cited on pages 62 and 79.)

[172] Maya Lincoln and Avigdor Gal. Searching Business Process Repositories Using Operational Similarity. In *On the Move to Meaningful Internet Systems*, volume 7044 of *Lecture Notes in Computer Science*, pages 2–19. Springer, 2011. (Cited on page 115.)

[173] Maya Lincoln, Mati Golani, and Avigdor Gal. Machine-Assisted Design of Business Process Models Using Descriptor Space Analysis. In *Business Process Management*, volume 6336 of *Lecture Notes in Computer Science*, pages 128–144. Springer, 2010. (Cited on page 109.)

[174] Alan Lipkus. A Proof of the Triangle Inequality for the Tanimoto Distance. *Journal of Mathematical Chemistry*, 26:263–265, 1999. (Cited on pages 154, 155, 167, and 172.)

[175] Ziyang Liu, Qihong Shao, and Yi Chen. Searching Workflows with Hierarchical Views. *Proc. VLDB Endow.*, 3(1-2):918–927, September 2010. (Cited on pages 109, 112, and 234.)

[176] Niels Lohmann. A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In *Web Services and Formal Methods*, volume 4937 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2008. (Cited on pages 72 and 128.)

[177] Niels Lohmann, Eric Verbeek, and Remco Dijkman. Petri Net Transformations for Business Processes — A Survey. In *Transactions on Petri Nets and Other Models of Concurrency*, volume 5460 of *Lecture Notes in Computer Science*, pages 46–63. Springer, 2009. (Cited on pages 39, 45, 55, 71, 72, 119, and 142.)

[178] Ruopeng Lu and Shazia Sadiq. On the Discovery of Preferred Work Practice Through Business Process Variants. In *Conceptual Modeling (ER)*, volume 4801 of *Lecture Notes in Computer Science*,

pages 165–180. Springer, 2007. (Cited on pages 113, 118, 143, 150, and 234.)

[179] Ruopeng Lu and Shazia Wasim Sadiq. Managing Process Variants as an Information Resource. In *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 426–431. Springer, 2006. (Cited on pages 109, 113, 118, 150, and 234.)

[180] Ruopeng Lu, Shazia Sadiq, and Guido Governatori. On Managing Business Processes Variants. *Data Knowl. Eng.*, 68(7):642–664, July 2009. (Cited on pages 96, 109, 113, 118, 136, 150, and 234.)

[181] Zhilei Ma, Wei Lu, and Frank Leymann. Query Structural Information of BPEL Processes. In *Internet and Web Applications and Services*, pages 401–406. IEEE Computer Society, 2009. (Cited on pages 40 and 148.)

[182] Jayant Madhavan, Philip A. Bernstein, Pedro Domingos, and Alon Y. Halevy. Representing and Reasoning about Mappings between Domain Models. In *Artificial intelligence*, pages 80–86. American Association for Artificial Intelligence, 2002. (Cited on pages 55, 57, and 58.)

[183] Therani Madhusudan, J. Leon Zhao, and Byron Marshall. A Case-Based Reasoning Framework for Workflow Model Management. *Data Knowl. Eng.*, 50:87–115, July 2004. (Cited on pages 146 and 237.)

[184] B. Mahleko and A. Wombacher. Indexing Business Processes based on Annotated Finite State Automata. In *Web Services*, pages 303–311. IEEE Computer Society Press, September 2006. (Cited on pages 109, 116, 117, 118, 149, 236, and 239.)

[185] Bendick Mahleko, Andreas Wombacher, and Peter Fankhauser. A Grammar-Based Index for Matching Business Processes. In *Web Services*, volume 1, pages 21–30. IEEE Computer Society Press, July 2005. (Cited on page 64.)

[186] Thomas W. Malone, Kevin Crowston, and George A. Herman. *Organizing Business Knowledge: The Mit Process Handbook*. Process handbook. MIT Press, 2003. (Cited on page 58.)

[187] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999. (Cited on pages 34, 37, and 58.)

[188] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge Uni-

versity Press, 1 edition, July 2008. (Cited on pages 62, 138, 142, and 180.)

[189] Selma Limam Mansar and Hajo A. Reijers. Best Practices in Business Process Redesign: Validation of a Redesign Framework. *Comput. Ind.*, 56:457–471, June 2005. (Cited on pages 74, 76, and 78.)

[190] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. ADDiff: Semantic Differencing for Activity Diagrams. In *Foundations of Software Sngineering*, pages 179–189. ACM, 2011. (Cited on pages 64, 143, 149, and 239.)

[191] Ivan Markovic. Advanced Querying and Reasoning on Business Process Models. In *Business Information Systems*, volume 7 of *Lecture Notes in Business Information Processing*, pages 189–200. Springer, 2008. (Cited on pages 36, 115, and 236.)

[192] Ivan Markovic and Alessandro Costa Pereira. Towards a Formal Framework for Reuse in Business Process Modeling. In *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 484–495. Springer, 2007. (Cited on pages 31, 39, 40, and 41.)

[193] Ivan Markovic, Alessandro Costa Pereira, David Francisco, and Henar Muñoz. Querying in Business Process Modeling. In *Service-Oriented Computing Workshops*, volume 4907 of *Lecture Notes in Computer Science*, pages 234–245. Springer, 2007. (Cited on pages 57, 109, 115, and 236.)

[194] Ivan Markovic, Alessandro C. Pereira, and Nenad Stojanovic. A Framework for Querying in Business Process Modelling. In *Multikonferenz Wirtschaftsinformatik*. GITO-Verlag, March 2008. (Cited on pages 32 and 40.)

[195] Laura Maruster, A. J. M. M. Weijters, Wil M. P. van der Aalst, and Antal van den Bosch. Process Mining: Discovering Direct Successors in Process Logs. In *Discovery Science*, volume 2534 of *Lecture Notes in Computer Science*, pages 364–373. Springer, 2002. (Cited on pages 8, 67, and 160.)

[196] Michael McGill, Matthew Koll, and Terry Noreault. *An Evaluation of Factors Affecting Document Ranking by Information Retrieval Systems*. Syracuse Univ., NY., 1979. (Cited on page 62.)

[197] Kenneth L. McMillan. A Technique of State Space Search Based on Unfolding. *Formal Methods in System Design*, 6:45–65, 1995. (Cited on pages 67 and 116.)

[198] George H. Mealy. A Method for Synthesizing Sequential Circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955. (Cited on page 63.)

[199] Kurt Mehlhorn. *Graph Algorithms and NP-Completeness*. Springer, 1984. (Cited on page 146.)

[200] Joachim Melcher and Detlef Seese. Visualization and Clustering of Business Process Collections Based on Process Metric Values. In *Symbolic and Numeric Algorithms for Scientific Computing*, pages 572–575. IEEE Computer Society, 2008. (Cited on pages 147 and 237.)

[201] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *Data Engineering*, pages 117–128. IEEE Computer Society, 2002. (Cited on page 146.)

[202] Jan Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. PhD thesis, Vienna University of Economics and Business Administration, May 2007. (Cited on pages 20, 24, and 147.)

[203] Jan Mendling. *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, volume 6 of *Lecture Notes in Business Information Processing*. Springer, 2008. (Cited on pages 88 and 184.)

[204] Jan Mendling and Carlo Simon. Business Process Design by View Integration. In *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 55–64. Springer, 2006. (Cited on pages 32 and 184.)

[205] Jan Mendling, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. On the Degree of Behavioral Similarity between Business Process Models. In *Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, volume 303 of *CEUR Workshop Proceedings*, pages 39–58. ceur-ws.org, 2007. (Cited on pages 67, 151, 153, and 240.)

[206] Bruno T. Messmer and Horst Bunke. A New Algorithm for Error-Tolerant Subgraph Isomorphism Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(5):493–504, May 1998. (Cited on page 113.)

[207] Bruno T. Messmer and Horst Bunke. Efficient Subgraph Isomorphism Detection: A Decomposition Approach. *IEEE Trans. on Knowl. and Data Eng.*, 12:307–323, March 2000. (Cited on pages 34 and 112.)

[208] Derek Miers. Best Practice (BPM). *ACM Queue*, 4(2):40–48, March 2006. (Cited on page 78.)

[209] Hafedh Mili, Guy Tremblay, Guitta Bou Jaoude, Éric Lefebvre, Lamia Elabed, and Ghizlane El Boussaidi. Business Process Modeling Languages: Sorting through the Alphabet Soup. *ACM Comput. Surv.*, 43(1):4:1–4:56, December 2010. (Cited on pages 22, 23, 24, 34, 71, 113, and 148.)

[210] George A. Miller. WordNet: A Lexical Database for English. *Commun. ACM*, 38:39–41, November 1995. (Cited on page 58.)

[211] Robin Milner. *A Calculus of Communicating Systems*. Springer, 1982. (Cited on pages 63, 65, and 146.)

[212] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes. *Information and Computation*, 100(1):1 – 40, 1992. (Cited on page 63.)

[213] Mirjam Minor, Alexander Tartakovski, and Ralph Bergmann. Representation and Structure-Based Similarity Assessment for Agile Workflows. In *Case-Based Reasoning Research and Development*, volume 4626 of *Lecture Notes in Computer Science*, pages 224–238. Springer, 2007. (Cited on pages 149, 200, and 238.)

[214] Michele Missikoff, Maurizio Proietti, and Fabrizio Smith. Querying Semantically Enriched Business Processes. In *Database and Expert Systems Applications*, volume 6861 of *Lecture Notes in Computer Science*, pages 294–302. Springer, 2011. (Cited on pages 109, 115, and 235.)

[215] Mariusz Momotko and Kazimierz Subieta. Process Query Language: A Way to Make Workflow Processes More Flexible. In *Advances in Databases and Information Systems*, volume 3255 of *Lecture Notes in Computer Science*, pages 306–321. Springer, 2004. (Cited on pages 108 and 115.)

[216] Edward F. Moore. Gedanken Experiments on Sequential Machines. In *Automata Studies*, pages 129–153. Princeton U., 1956. (Cited on page 63.)

[217] Jens Müller. A Rule-Based Approach to Match Structural Patterns with Business Process Models. In *Rule Interchange and Applications*, volume 5858 of *Lecture Notes in Computer Science*, pages 208–215. Springer, 2009. (Cited on pages 108, 109, 114, and 235.)

[218] Tadao Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989. (Cited on pages 45 and 72.)

[219] Gonzalo Navarro. A Guided Tour to Approximate String Matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001. (Cited on page 58.)

[220] Roberto Navigli. Word Sense Disambiguation: A Survey. *ACM Comput. Surv.*, 41(2):10:1–10:69, February 2009. (Cited on page 57.)

[221] Shiva Nejati, Mehrdad Sabetzadeh, Marsha Chechik, Steve Easterbrook, and Pamela Zave. Matching and Merging of Statecharts Specifications. In *Software Engineering*, pages 54–64. IEEE Computer Society, 2007. (Cited on pages 32, 34, 58, 66, and 144.)

[222] Florian Niedermann, Sylvia Radeschutz, and Bernhard Mitschang. Design-Time Process Optimization through Optimization Patterns and Process Model Matching. In *Commerce and Enterprise Computing*, pages 48–55. IEEE Computer Society, 2010. (Cited on page 58.)

[223] Michael Niemann, Melanie Siebenhaar, Stefan Schulte, and Ralf Steinmetz. Comparison and Retrieval of Process Models using Related Cluster Pairs. *Comput. Ind.*, 63(2):168–180, February 2012. (Cited on pages 148, 157, and 238.)

[224] Anil Nigam and Nathan S. Caswell. Business Artifacts: An Approach to Operational Specification. *IBM Systems Journal*, 42 (3):428–445, 2003. (Cited on pages 24 and 71.)

[225] Object Management Group. Business Process Model and Notation (BPMN) Specification, Version 2.0, 2009. (Cited on pages 18, 24, 25, 71, 72, and 185.)

[226] Avner Ottensooser, Alan Fekete, Hajo A. Reijers, Jan Mendling, and Con Menictas. Making Sense of Business Process Descriptions: An Experimental Comparison of Graphical and Textual Notations. *J. Syst. Softw.*, 85(3):596–606, March 2012. (Cited on page 24.)

[227] Chun Ouyang, Eric Verbeek, Wil M. P. van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur H. M. ter Hofstede. Formal Semantics and Analysis of Control Flow in WS-BPEL. *Sci. Comput. Program.*, 67(2-3):162–198, July 2007. (Cited on page 72.)

[228] Chun Ouyang, Arthur H.M. ter Hofstede, Marcello La Rosa, Liang Song, Jianmin Wang, and Artem Polyvyanyy. APQL: A Process-Model Query Language. Technical report, Queensland University of Technology, 2013. (Cited on pages 108, 116, 126, and 236.)

[229] Victor Pankratius and Wolffried Stucky. A Formal Foundation for Workflow Composition, Workflow View Definition,

and Workflow Normalization Based on Petri Nets. In *Asia-Pacific Conference on Conceptual Modelling*, pages 79–88. Australian Computer Society, Inc., 2005. (Cited on pages 113 and 234.)

[230] David Park. Concurrency and Automata on Infinite Sequences. In *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981. (Cited on page 65.)

[231] Maja Pesic and Wil M. P. van der Aalst. A Declarative Approach for Flexible Business Processes Management. In *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2006. (Cited on pages 24 and 71.)

[232] Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Bonn, Institut für Instrumentelle Mathematik, 1962. (Cited on pages 45 and 63.)

[233] Amir Pnueli. The Temporal Logic of Programs. In *Foundations of Computer Science*, pages 46–57. IEEE Computer Society, 1977. (Cited on page 116.)

[234] Artem Polyvyanyy, Sergey Smirnov, and Mathias Weske. Process Model Abstraction: A Slider Approach. In *Enterprise Distributed Object Computing Conference*, pages 325–331. IEEE Computer Society, 2008. (Cited on page 27.)

[235] Artem Polyvyanyy, Sergey Smirnov, and Mathias Weske. Business Process Model Abstraction. In *Handbook on Business Process Management 1*, International Handbooks on Information Systems, pages 149–166. Springer, 2010. (Cited on page 26.)

[236] Lucia Pomello, Grzegorz Rozenberg, and Carla Simone. A Survey of Equivalence Notions for Net Based Systems. In *Advances in Petri Nets*, volume 609 of *Lecture Notes in Computer Science*, pages 410–472. Springer, 1992. (Cited on page 63.)

[237] Martin F. Porter. An Algorithm for Suffix Stripping. In *Readings in Information Retrieval*, pages 313–316. Morgan Kaufmann Publishers Inc., 1997. (Cited on page 58.)

[238] Michael E. Porter. *Competitive Advantage: Creating and Sustaining Superior Performance*. Free Press, 1985. (Cited on page 18.)

[239] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`, January 2008. (Cited on page 115.)

[240] Mu Qiao, Rama Akkiraju, and Aubrey J. Rembert. Towards Efficient Business Process Clustering and Retrieval: Combining Language Modeling and Structure Matching. In *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 199–214. Springer, 2011. (Cited on pages 31, 144, and 149.)

[241] Ivo Raedts, Marija Petkovic, Yaroslav S. Usenko, Jan Martijn E. M. van der Werf, Jan Friso Groote, and Lou J. Somers. Transformation of BPMN Models for Behaviour Analysis. In *Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 126–137. INSTICC Press, 2007. (Cited on page 72.)

[242] Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10(4):334–350, 2001. (Cited on pages 39, 54, and 82.)

[243] Hajo A. Reijers. *Design and Control of Workflow Processes: Business Process Management for the Service Industry*. Springer, 2003. (Cited on page 16.)

[244] Hajo A. Reijers and S. Liman Mansar. Best Practices in Business Process Redesign: An Overview and Qualitative Evaluation of Successful Redesign Heuristics. *Omega*, 33(4):283 – 306, 2005. (Cited on page 27.)

[245] Hajo A. Reijers, Thomas Freytag, Jan Mendling, and Andreas Eckleder. Syntax Highlighting in Business Process Models. *Decis. Support Syst.*, 51(3):339–349, June 2011. (Cited on page 96.)

[246] Wolfgang Reisig. *Petri Nets: An Introduction*. Springer, 1985. (Cited on page 45.)

[247] Naphtali Rishe, Jun Yuan, Rukshan Athauda, Shu-Ching Chen, Xiaoling Lu, Xiaobin Ma, Alexander Vaschillo, Artyom Shaposhnikov, and Dmitry Vasilevsky. Semantic Access: Semantic Interface for Querying Databases. In *Very Large Data Bases*, pages 591–594. Morgan Kaufmann, 2000. (Cited on page 54.)

[248] Andreas Rogge-Solti and Mathias Weske. Enabling Probabilistic Process Monitoring in Non-automated Environments. In *Enterprise, Business-Process and Information Systems Modeling*, volume 113 of *Lecture Notes in Business Information Processing*, pages 226–240. Springer, 2012. (Cited on page 28.)

[249] Marcello La Rosa, Marlon Dumas, Reina Uba, and Remco M. Dijkman. Business Process Model Merging : An Approach to Business Process Consolidation. *ACM Transactions on Software Engineering and Methodology*, 2012. (Cited on pages 32, 143, and 184.)

[250] Michael Rosemann. Potential Pitfalls of Process Modeling: Part B. *Business Process Management Journal*, 12(3):377–384, 2006. (Cited on pages 29 and 32.)

[251] Nick Russell, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and David Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In *Advanced Information Systems Engineering*, volume 3520 of *Lecture Notes in Computer Science*, pages 216–232. Springer, 2005. (Cited on pages 23 and 29.)

[252] Tomer Sagi and Avigdor Gal. Non-binary Evaluation for Schema Matching. In *Conceptual Modeling (ER)*, volume 7532 of *Lecture Notes in Computer Science*, pages 477–486. Springer, 2012. (Cited on page 58.)

[253] Sherif Sakr and Ghazi Al-Naymat. Graph Indexing and Querying: A Review. *International Journal of Web Information Systems*, 6(2):101–120, 2010. (Cited on page 135.)

[254] Sherif Sakr, Ahmed Awad, and Matthias Kunze. Querying Process Models Repositories by Aggregated Graph Search. In *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 573–585. Springer, 2012. (Cited on pages 9, 106, and 109.)

[255] Gerard Salton. *Automatic Information Organization and Retrieval.* McGraw Hill Text, 1968. (Cited on pages 67 and 147.)

[256] Gerard M. Salton, Andrew Wong, and Chung S. Yang. A Vector Space Model for Automatic Indexing. *Commun. ACM*, 18(11): 613–620, 1975. (Cited on pages 58, 146, and 147.)

[257] August-Wilhelm Scheer and Markus Nüttgens. ARIS Architecture and Reference Models for Business Process Management. In *Business Process Management*, volume 1806 of *Lecture Notes in Computer Science*, pages 376–389. Springer, 2000. (Cited on page 17.)

[258] Khurram Shahzad, Birger Andersson, Maria Bergholtz, Ananda Edirisuriya, Tharaka Ilayperuma, Prasad Jayaweera, and Paul Johannesson. Elicitation of Requirements for a Business Process Model Repository. In *Business Process Management Workshops*, volume 17 of *Lecture Notes in Business Information Processing*, pages 44–55. Springer, 2009. (Cited on pages 23 and 30.)

[259] Qihong Shao, Peng Sun, and Yi Chen. WISE: A Workflow Information Search Engine. In *Data Engineering*, pages 1491–1494. IEEE Computer Society, 2009. (Cited on pages 112 and 234.)

[260] Dennis Shasha, Jason T. L. Wang, and Rosalba Giugno. Algorithmics and Applications of Tree and Graph Searching. In *Principles of Database Systems*, pages 39–52. ACM, 2002. (Cited on page 135.)

[261] Zhongnan Shen and Jianwen Su. Web Service Discovery Based on Behavior Signatures. In *Services Computing*, pages 279–286. IEEE Computer Society, 2005. (Cited on pages 109, 116, and 236.)

[262] Pavel Shvaiko and Jérôme Euzenat. A Survey of Schema-Based Matching Approaches. In *Journal on Data Semantics*, volume 3730 of *Lecture Notes in Computer Science*, pages 146–171. Springer, 2005. (Cited on pages 39 and 54.)

[263] Jim Sinur and Janelle Hill. Magic Quadrant for Business Process Management Suites. Technical report, Gartner Research, October 2010. (Cited on page 78.)

[264] Sergey Smirnov, Matthias Weidlich, Jan Mendling, and Mathias Weske. Action Patterns in Business Process Model Repositories. *Computers in Industry*, 63(2):98–111, 2012. (Cited on page 184.)

[265] Fabrizio Smith, Michele Missikoff, and Maurizio Proietti. Ontology-Based Querying of Composite Services. In *Business System Management and Engineering*, volume 7350 of *Lecture Notes in Computer Science*, pages 159–180. Springer, 2012. (Cited on pages 33, 57, 109, 115, and 235.)

[266] Howard Smith and Peter Fingar. *Business Process Management: The Third Wave*. Meghan-Kiffer Press, 2006. (Cited on pages 4, 16, 17, and 27.)

[267] Oleg Sokolsky, Sampath Kannan, and Insup Lee. Simulation-Based Graph Similarity. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *Lecture Notes in Computer Science*, pages 426–440. Springer, 2006. (Cited on pages 34 and 66.)

[268] Liang Song, Jianmin Wang, Lijie Wen, Wenxing Wang, Shijie Tan, and Hui Kong. Querying Process Models Based on the Temporal Relations between Tasks. In *Enterprise Distributed Object Computing Conference Workshops*, pages 213–222. IEEE Computer Society, 2011. (Cited on pages 108, 109, 116, 117, 126, and 236.)

[269] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer, 1973. (Cited on page 22.)

[270] Tony Spiteri Staines. Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams

and Colored Petri Nets. In *Engineering of Computer Based Systems*, pages 191–200. IEEE Computer Society, 2008. (Cited on page 72.)

[271] Ping Sun. Service Clustering Based on Profile and Process Similarity. In *Information Science and Engineering*, pages 535–539. IEEE Computer Society, 2010. (Cited on pages 64, 65, 74, 143, 151, 153, 154, and 239.)

[272] Tsukasa Takemura. Formal Semantics and Verification of BPMN Transaction and Compensation. In *Asia-Pacific Services Computing*, pages 284–290. IEEE Computer Society, 2008. (Cited on page 72.)

[273] Ivana Trickovic. Formalizing Activity Diagram of UML by Petri Nets. *Novi Sad Journal of Mathematics*, 30(3):161–171, 2000. (Cited on page 72.)

[274] Amos Tversky. Features of Similarity. In *Psychological Review*, volume 84, pages 327–352, 1977. (Cited on pages 79, 81, 108, and 143.)

[275] Reina Uba, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Clone Detection in Repositories of Business Process Models. In *BPM*, volume 6896 of *Lecture Notes in Computer Science*, pages 248–264. Springer, 2011. (Cited on pages 109, 113, 117, 143, 154, and 234.)

[276] Jeffrey K. Uhlmann. Metric Trees. *Applied Mathematics Letters*, 4: 61–62, 1991. (Cited on pages 37, 87, 135, 154, 165, and 166.)

[277] Jeffrey K. Uhlmann. Satisfying General Proximity/Similarity Queries with Metric Trees. *Information Processing Letters*, 40(4): 175–179, 1991. (Cited on pages 167 and 168.)

[278] Julian R. Ullmann. An Algorithm for Subgraph Isomorphism. *J. ACM*, 23(1):31–42, January 1976. (Cited on page 112.)

[279] Antti Valmari. Stubborn Sets for Reduced State Space Generation. In *Applications and Theory of Petri Nets: Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer, 1991. (Cited on page 105.)

[280] Antti Valmari. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer, 1998. (Cited on pages 47, 66, 70, 76, 105, 123, 132, 151, and 155.)

[281] Wil M. M. P.P. van der Aalst. Inheritance of Business Processes: A Journey Visiting Four Notorious Problems. In *Petri Net Technology for Communication-Based Systems*, volume 2472 of *Lecture*

Notes in Computer Science, pages 383–408. Springer, 2003. (Cited on pages 116 and 118.)

[282] Wil M. P. van der Aalst. Verification of Workflow Nets. In *Application and Theory of Petri Nets*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer, 1997. (Cited on pages 26, 48, 49, 50, 73, 79, and 97.)

[283] Wil M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998. (Cited on pages 8, 45, 50, 55, 63, 71, 102, and 103.)

[284] Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011. (Cited on pages 52, 67, and 160.)

[285] Wil M. P. van der Aalst. A Decade of Business Process Management Conferences: Personal Reflections on a Developing Discipline. In *Business Process Management*, volume 7481 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2012. (Cited on page 24.)

[286] Wil M. P. van der Aalst and Twan Basten. Identifying Commonalities and Differences in Object Life Cycles Using Behavioral Inheritance. In *Application and Theory of Petri Nets*, volume 2075 of *Lecture Notes in Computer Science*, pages 32–52. Springer, 2001. (Cited on page 65.)

[287] Wil M. P. van der Aalst and Twan Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theor. Comput. Sci.*, 270(1-2):125–203, 2002. (Cited on pages 65 and 146.)

[288] Wil M. P. van der Aalst and Kees van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002. (Cited on pages 16, 18, and 20.)

[289] Wil M. P. Van Der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske. Business Process Management: A Survey. In *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003. (Cited on pages 18, 20, 26, and 27.)

[290] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow Patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003. (Cited on pages 23, 74, 127, and 211.)

[291] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow Mining: Discovering Process Models from Event

Logs. *IEEE Trans. on Knowl. and Data Eng.*, 16:1128–1142, September 2004. (Cited on pages 52, 67, and 68.)

[292] Wil M. P. van der Aalst, Mathias Weske, and Dolf Grünbauer. Case Handling: A New Paradigm for Business Process Support. *Data and Knowledge Engineering*, 53:2005, 2005. (Cited on pages 24 and 71.)

[293] Wil M. P. van der Aalst, Ana K. A. de Medeiros, and A. J. M. M. Weijters. Process Equivalence: Comparing Two Process Models Based on Observed Behavior. In *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2006. (Cited on pages 147, 150, 153, 237, and 239.)

[294] Wil M. P. van der Aalst, A. Dreiling, F. Gottschalk, M. Rosemann, and M.H. Jansen-Vullers. Configurable Process Models as a Basis for Reference Modeling. In *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 512–518. Springer, 2006. (Cited on pages 32 and 144.)

[295] Wil M. P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysis. Technical report, Eindhoven University of Technology, 2008. (Cited on page 51.)

[296] Wil M. P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative Workflows: Balancing between Flexibility and Support. *Computer Science - Research and Development*, 23:99–113, 2009. (Cited on pages 24 and 71.)

[297] Boudewijn F. van Dongen, Ana K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and Wil M. P. van der Aalst. The ProM Framework: A New Era in Process Mining Tool Support. In *Application and Theory of Petri Nets*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2005. (Cited on page 72.)

[298] Boudewijn. F. van Dongen, Jan Mendling, and Wil M. P. van Aalst. Structural Patterns for Soundness of Business Process Models. In *Enterprise Distributed Object Computing*, pages 116–128. IEEE Computer Society, 2006. (Cited on pages 34 and 66.)

[299] Boudewijn F. van Dongen, Monique H. Jansen-Vullers, Eric H. M. W. Verbeek, and Wil M. P. van der Aalst. Verification of the SAP Reference Models using EPC Reduction, State Space Analysis, and Invariants. *Comput. Ind.*, 58(6):578–601, August 2007. (Cited on pages 72, 128, and 184.)

[300] Boudewijn F. van Dongen, Remco Dijkman, and Jan Mendling. Measuring Similarity between Business Process Models. In *Advanced Information Systems Engineering*, volume 5074 of *Lecture Notes in Computer Science*, pages 450–464. Springer, 2008. (Cited on pages 34, 58, 63, 67, 142, 143, 151, 153, 154, 157, and 240.)

[301] Rob J. van Glabbeek. The Linear Time-Branching Time Spectrum. In *Theories of Concurrency: Unification and Extension*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990. (Cited on pages 63, 65, 78, and 146.)

[302] Rob J. van Glabbeek. The Linear Time-Branching Time Spectrum II. In *Theories of Concurrency: Unification and Extension*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993. (Cited on pages 65 and 116.)

[303] Rob J. van Glabbeek and W. Peter Weijland. Branching Time and Abstraction in Bisimulation Semantics. *J. ACM*, 43(3):555–600, 1996. (Cited on pages 62 and 65.)

[304] Cornelis J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979. (Cited on pages 83 and 178.)

[305] Irene Vanderfeesten, Jorge Cardoso, Hajo A. Reijers, and Wil M. P. van der Aalst. Quality Metrics for Business Process Models. *BPM and Workflow handbook*, pages 1–12, 2006. (Cited on page 88.)

[306] Jussi Vanhatalo, Hagen Völzer, Frank Leymann, and Simon Moser. Automatic Workflow Graph Refactoring and Completion. In *Service-Oriented Computing*, volume 5364 of *Lecture Notes in Computer Science*, pages 100–115. Springer, 2008. (Cited on pages 50 and 72.)

[307] Jianmin Wang, Tengfei He, Lijie Wen, Nianhua Wu, Arthur H. M. ter Hofstede, and Jianwen Su. A Behavioral Similarity Measure between Labeled Petri Nets Based on Principal Transition Sequences. In *On the Move to Meaningful Internet Systems*, volume 6426 of *Lecture Notes in Computer Science*, pages 394–401. Springer, 2010. (Cited on pages 65, 150, 153, and 239.)

[308] Jianmin Wang, Tao Jin, RaymondK. Wong, and Lijie Wen. Querying Business Process Model Repositories. *World Wide Web*, pages 1–28, 2013. (Cited on pages 37 and 68.)

[309] Richard Y. Wang and Diane M. Strong. Beyond Accuracy: What Data Quality Means to Data Consumers. *J. Manage. Inf. Syst.*, 12: 5–33, March 1996. (Cited on page 88.)

[310] Yiqiao Wang and Eleni Stroulia. Flexible Interface Matching for Web-Service Discovery. In *Web Information Systems Engineering*, pages 147–156. IEEE Computer Society, 2003. (Cited on page 54.)

[311] Barbara Weber and Manfred Reichert. Refactoring Process Models in Large Process Repositories. In *Advanced Information Systems Engineering*, volume 5074 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2008. (Cited on pages 31, 64, and 106.)

[312] Matthias Weidlich. *Behavioral Profiles – A Relational Approach to Behaviour Consistency*. PhD thesis, Hasso Plattner Institute, University of Potsdam, 2011. (Cited on pages 55 and 77.)

[313] Matthias Weidlich and Jan Mendling. Perceived Consistency between Process Models. *Information Systems*, 37(2):80–98, April 2012. (Cited on pages 74, 79, and 155.)

[314] Matthias Weidlich and Jan Martijn Werf. On Profiles and Footprints – Relational Semantics for Petri Nets. In *Application and Theory of Petri Nets*, volume 7347 of *Lecture Notes in Computer Science*, pages 148–167. Springer, 2012. (Cited on pages 8, 34, 51, 52, 53, 67, 68, 73, 74, 76, 78, 80, 81, 102, 126, 133, 144, 153, and 160.)

[315] Matthias Weidlich, Mathias Weske, and Jan Mendling. Change propagation in process models using behavioural profiles. In *High Performance Computing, Networking Storage and Analysis*, pages 33–40. IEEE Computer Society, 2009. (Cited on pages 8, 67, 68, 116, and 160.)

[316] Matthias Weidlich, Remco Dijkman, and Jan Mendling. The ICoP Framework: Identification of Correspondences between Process Models. In *Advanced Information Systems Engineering*, volume 6051 of *Lecture Notes in Computer Science*, pages 483–498. Springer-Verlag, 2010. (Cited on pages 8, 39, 54, 55, 56, 69, 82, 83, 146, and 212.)

[317] Matthias Weidlich, Felix Elliger, and Mathias Weske. Generalised Computation of Behavioural Profiles Based on Petri-Net Unfoldings. In *Web Services and Formal Methods*, volume 6551 of *Lecture Notes in Computer Science*, pages 101–115. Springer, 2010. (Cited on pages 53, 67, and 73.)

[318] Matthias Weidlich, Artem Polyvyanyy, Jan Mendling, and Mathias Weske. Efficient Computation of Causal Behavioural Profiles Using Structural Decomposition. In *Application and Theory of Petri Nets*, volume 6128 of *Lecture Notes in Computer*

Science, pages 63–83. Springer, 2010. (Cited on pages 68, 73, 78, and 214.)

[319] Matthias Weidlich, Jan Mendling, and Mathias Weske. Efficient Consistency Measurement Based on Behavioral Profiles of Process Models. *IEEE Trans. Softw. Eng.*, 37(3):410–429, May 2011. (Cited on pages 52, 67, 73, 74, 84, 103, 125, 152, 155, 160, and 161.)

[320] Matthias Weidlich, Jan Mendling, and Mathias Weske. A Foundational Approach for Managing Process Variability. In *Advanced Information Systems Engineering*, volume 6741 of *Lecture Notes in Computer Science*, pages 267–282. Springer, 2011. (Cited on page 162.)

[321] Mathias Weske. *Workflow Management Systems: Formal Foundation, Conceptual Design, Implementation Aspects*. PhD thesis, University of Münster, 2000. (Cited on page 63.)

[322] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2nd edition edition, June 2012. (Cited on pages 16, 18, 19, 20, 21, 22, 26, 29, 55, 62, 143, and 146.)

[323] Andreas Wombacher. Evaluation of Technical Measures for Workflow Similarity Based on a Pilot Study. In *On the Move to Meaningful Internet Systems*, volume 4275 of *Lecture Notes in Computer Science*, pages 255–272. Springer, 2006. (Cited on pages 65, 116, 117, 118, 151, 153, 154, 236, and 239.)

[324] Andreas Wombacher and Chen Li. Alternative Approaches for Workflow Similarity. In *High Performance Computing, Networking Storage and Analysis*, pages 337–345. IEEE Computer Society, 2010. (Cited on pages 65, 151, 153, 154, and 239.)

[325] Andreas Wombacher, Peter Fankhauser, Bendick Mahleko, and Erich Neuhold. Matchmaking for Business Processes. In *E-Commerce Technology*, page 7. IEEE Computer Society, 2003. (Cited on pages 149 and 239.)

[326] Lan Xiao, Li Zheng, Jian Xiao, and Yi Huang. A Graphical Query Language for Querying Petri Nets. In *Information Systems: Modeling, Development, and Integration*, volume 20 of *Lecture Notes in Business Information Processing*, pages 514–525. Springer, 2009. (Cited on pages 114 and 235.)

[327] Xifeng Yan, Philip S. Yu, and Jiawei Han. Graph Indexing: A Frequent Structure-Based Approach. In *Management of Data*, pages 335–346. ACM, 2004. (Cited on page 135.)

[328] Zhiqiang Yan, Remco Dijkman, and Paul Grefen. Fast Business Process Similarity Search with Feature-Based Similarity Estimation. In *On the Move to Meaningful Internet Systems*, volume 6426 of *Lecture Notes in Computer Science*, pages 60–77. Springer, 2010. (Cited on pages 34, 148, 184, and 238.)

[329] Zhiqiang Yan, Remco Dijkman, and Paul Grefen. Fast Business Process Similarity Search. *Distributed and Parallel Databases*, 30: 105–144, 2012. (Cited on pages 136, 148, 154, and 238.)

[330] Zhiqiang Yan, Remco Dijkman, and Paul Grefen. FNet: An Index for Advanced Business Process Querying. In *Business Process Management*, volume 7481 of *Lecture Notes in Computer Science*, pages 246–261. Springer, 2012. (Cited on pages 114, 117, and 235.)

[331] Zhiqiang Yan, Remco M. Dijkman, and Paul W. P. J. Grefen. Business Process Model Repositories - Framework and Survey. *Information & Software Technology*, 54(4):380–395, 2012. (Cited on pages 4, 23, 30, 109, and 112.)

[332] Peter N. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Discrete algorithms*, pages 311–321. Society for Industrial and Applied Mathematics, 1993. (Cited on page 168.)

[333] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach*. Springer, 2005. (Cited on pages 37, 86, 87, 108, 142, 143, 151, and 167.)

[334] Haiping Zha, Jianmin Wang, Lijie Wen, Chaokun Wang, and Jiaguang Sun. A Workflow Net Similarity Measure Based on Transition Adjacency Relations. *Computers in Industry*, 61(5): 463–471, 2010. (Cited on pages 67, 143, 150, 151, 153, 154, 239, and 240.)

[335] Hai Zhuge. A Process Matching Approach for Flexible Workflow Process Reuse. *Information & Software Technology*, 44(8): 445–450, 2002. (Cited on page 109.)

[336] Sergiy Zlatkin and Roland Kaschek. Towards Amplifying Business Process Reuse. In *Perspectives in Conceptual Modeling*, volume 3770 of *Lecture Notes in Computer Science*, pages 364–374. Springer, 2005. (Cited on page 32.)

[337] Moshé M. Zloof. Query by Example. In *National Computer Conference and Exposition*, pages 431–438. ACM, 1975. (Cited on pages 70, 110, and 118.)

I have to follow my thoughts and mine for the gold.
I have to dig it out.

— William Henry Cosby, Jr.

ACKNOWLEDGMENTS

Each book begins with an empty page, each painting on a white canvas. When I started writing the manuscript for this thesis, I was looking at a blank computer screen, the cursor silently blinking. I knew what I wanted to say, but couldn't quite figure out how to get there. As I write these lines, I have gotten there. And while I already cannot remember anymore, how I actually got started, I know that many friends have their stake in the result.

I deeply thank Mathias Weske, my supervisor, for his continuous encouragement and support throughout the last four years. His guidance and the freedom he entrusted me with made it possible for me to follow my thoughts and to pursue my own research interest. Moreover, he gave me the amazing opportunity to participate in teaching. More than anything else, this equipped me with scientific skills and practical views on my research that would otherwise not have been possible.

Contents of this thesis has been inspired and shaped by countless discussions with many colleagues and friends, among them my reviewers Jan Mendling and Hajo Reijers who provided valuable feedback and shaped my perspective on this research. I owe gratitude to Matthias Weidlich, with whom I seeded the idea of searching process models on the basis of behavioral relations. His ongoing enthusiasm and encouragement, and many insightful discussions helped me to grow this idea further. I am grateful to Dirk Fahland, who offered invaluable advice to overcome the hardship of writing a thesis and helped me shape my ideas. It is due to the insights he shared with me that I managed to submit this thesis within my own schedule.

My gratitude extends to my colleagues and former colleagues at the HPI and the BPT group for a comfortable and inspiring work atmosphere. In many challenging and inspiring meetings, we discussed and improved details of my research, but also enjoyed not so scientific exchanges of ideas.

Besonderer Dank gilt meiner Familie für ihren unermüdlichen Rückhalt. Meinen Eltern verdanke ich vor allem meine Neugier für das Unergründete und den Mut jenes zu ergründen. Ohne vorauszugehen zeigten sie mir stets Wege in und aus Lebenslagen und unterstützten bedingungslos alle meine Vorhaben, selbst wenn sie mich in die Ferne trugen. Marietta und Annabelle gebührt mein innigster Dank für

ihre Geduld, Ausdauer und Verständnis für meine Arbeit. Ohne euch wäre ich sicher an meinem Schreibtisch festgewachsen, und ihr gabt mir die Liebe, Geborgenheit und Kraft, diese Arbeit abzuschließen.

Matthias Kunze
June, 2013