Business Process Model
Repositories - Efficient
Process Retrieval
Yan, Zhiqiang


A catalogue record is available from the Eindhoven University of Technology Library

ISBN: 978-90-386-3227-8
NUR: 982
Keywords: Business Process Model / Repository / Process Retrieval / Similarity Search / Querying / Indexing

The research described in this thesis has been carried out under the auspices of Beta Research School for Operations Management and Logistics. Beta Dissertation Series D159.

The research has been sponsored by the China Scholarship Council (CSC).

Printed by Printservice, Technische Universiteit Eindhoven
Cover design: Mingshuo Zhang

Copyright © 2012 by Zhiqiang Yan. All Rights Reserved.
Business Process Model Repositories -
Efficient Process Retrieval

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus, prof.dr.ir. C.J. van Duijn, voor een commissie aangewezen door het College voor Promoties in het openbaar te verdedigen op maandag 25 september 2012 om 14.00 uur

door

Zhiqiang Yan

geboren te Liaoning, China
Dit proefschrift is goedgekeurd door de promotor:

prof.dr.ir. P.W.P.J. Grefen

Co-promotor:
dr.ir. R.M. Dijkman
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.3.8 Oryx</td>
<td>163</td>
</tr>
<tr>
<td>A.3.9 BP-Suite</td>
<td>164</td>
</tr>
<tr>
<td>A.3.10 ProcessGene</td>
<td>164</td>
</tr>
<tr>
<td>A.3.11 The Process Variants Repository</td>
<td>165</td>
</tr>
<tr>
<td>A.3.12 The Querying Framework</td>
<td>165</td>
</tr>
<tr>
<td>A.3.13 The Semantic Business Process Repository</td>
<td>166</td>
</tr>
<tr>
<td>A.3.14 The BPEL Repository</td>
<td>166</td>
</tr>
<tr>
<td>A.3.15 Prosero</td>
<td>166</td>
</tr>
<tr>
<td>A.3.16 OSIRIS</td>
<td>167</td>
</tr>
<tr>
<td>A.3.17 APROMORE</td>
<td>167</td>
</tr>
<tr>
<td>A.3.18 Fragmento</td>
<td>168</td>
</tr>
<tr>
<td>A.3.19 BPCM</td>
<td>168</td>
</tr>
<tr>
<td>A.3.20 IPR</td>
<td>168</td>
</tr>
</tbody>
</table>

Summary | 171 |

Acknowledgements | 173 |

Curriculum Vitae | 175 |
Part I

Opening
Chapter 1

Introduction

1.1 Research Context and Motivation

As it becomes more common for organizations to describe their operations in terms of business processes, collections of business process models grow to contain hundreds or even thousands of business process models. For example, the SAP reference model collection contains over 600 business process models \cite{27} and a collection of business process models for Dutch local government contains a similar number of business process models \cite{36}. Managing such complex process landscapes is a difficult task. Typical issues arise, like: being able to find a particular process in a collection, managing different versions of processes and maintaining consistency when multiple people are editing the same process at the same time. In addition to that, the availability of a large collection of processes opens up new possibilities, like: extracting knowledge about the operations of the organization from the collection or re-using (best-practice) process fragments from the collection to design new processes.

As a reaction, software tools \cite{25,41} have been developed to help perform such tasks. These tools have been built as extensions of general database and repository systems. However, they have been specialized for storing business process models by using conceptual models, for example database schemas, that are process specific and by defining process specific interfaces. The interface could, for example, take the form of a Web service interface or an API that has operations like...
Chapter 1. Introduction

‘addProcess’ and ‘searchTask’ and through which process models can be imported or exported in process-specific interchange formats like EPML or PNML \[67\]. We refer to such repositories as Business Process (BP) Model Repositories, which we define as repositories that are structured according to a process-specific conceptual model and/or that have a process-specific interface. In addition to exploiting the functionality that is commonly provided by repository and database management systems \[19, 77\], BP Model Repositories provide functionality that is specific for repositories that contain business process models. Examples of process specific functionality include: functionality to assist with lifecycle management of business processes, functionality to help maintain consistency between the private view on business processes (which is the view that organizations have internally on their business processes) and the public view on business processes (which is the view on those parts of business processes that companies want to make visible publicly), and functionality to assist with configuration management of business processes as they are composed of (certain versions of) sub-processes and tasks.

1.2 Research Goal

The research goal of the thesis is to enhance BP Model Repository technology for large business model collections in a relevant aspect that has received little attention so far. To reach the goal, a relevant aspect needs to be selected, which can be done by analyzing functionality of existing BP Model Repositories and identifying the functionality that is neglected by existing BP Model Repositories. Looking ahead, the selected relevant aspect is efficient process retrieval. The aspect is identified as a relevant aspect of BP Model Repository technology by Becker et al. \[13\] and Dijkman et al. \[33\]; the aspect is selected because seventeen out of twenty BP Model Repositories we have analyzed support process retrieval, but none of them provides a technique to make retrieval efficient. Therefore, techniques need to be proposed to enhance the efficiency of process retrieval, making it at least an order of magnitude faster than current techniques.

In summary, the research goal can be divided into the following subgoals.

- Subgoal 1: Analyzing the state of the art of the BP Model Repository technology to identify an unsolved aspect, i.e., efficient process retrieval.
1.3. Research Approach

- Subgoal 2: Elaborating, implementing, and evaluating efficient process retrieval.
- Subgoal 3: Evaluating the addition of efficient process retrieval to the state of the art of the BP Model Repository technology.

1.3 Research Approach

Section 1.2 describes the goal (and subgoals) of the thesis. In this section, the research approach to reach the goal (and subgoals) is presented. The approach consists of nine steps and each step answers a research question.

The following questions are to be answered in the thesis.

- Question 1: What functionality is supported by existing BP Model Repositories?
- Question 2: Why is efficient process retrieval the aspect to be enhanced?
- Question 3: What techniques have been proposed to perform process retrieval?
- Question 4: How to enhance the efficiency of process retrieval?
- Question 5: How to integrate the proposed technique with process retrieval?
- Question 6: How to implement a prototype for the proposed process retrieval techniques?
- Question 7: How to evaluate the efficiency of the proposed process retrieval techniques?
- Question 8: What is the contribution of the proposed techniques to the state of the art of BP Model Repository technology?

To answer these questions, the following steps are performed. Each question is answered by one step.

- Step 1: Summarizing functionality supported by existing BP Model Repositories through a literature survey.
- Step 2: Observing the existing BP Model Repositories, surveyed in step 1, and selecting the aspect to enhance.
- Step 3: Summarizing existing techniques for process retrieval through a literature survey.
Chapter 1. Introduction

- Step 4: Observing the existing process retrieval techniques, surveyed in step 3, and proposing a technique for enhancing their efficiency.
- Step 5: Integrating the proposed technique with process retrieval techniques.
- Step 6: Designing and implementing a prototype for the proposed process retrieval techniques.
- Step 7: Running experiments to evaluate the efficiency of the proposed process retrieval techniques.
- Step 8: Evaluating the contribution of the proposed process retrieval techniques in the perspective of the BP Model Repository technology.

1.4 Deliverables and Contributions

Section 1.3 presents the approach of the thesis in terms of steps. This section presents the outputs of these steps in terms of deliverables and contributions.

Each of the eight steps has a deliverable. The deliverables are listed as follows.

- Deliverable 2: A survey of BP Model Repositories and the selected aspect to enhance.
- Deliverable 3: An overview of existing process retrieval techniques.
- Deliverable 4: Specification of a technique for enhancing the efficiency of process retrieval.
- Deliverable 5: Specification of efficient process retrieval techniques.
- Deliverable 6: A prototype for the proposed process retrieval techniques.
- Deliverable 7: Evaluation results of the proposed process retrieval techniques.
- Deliverable 8: Evaluation results of the addition of the proposed process retrieval techniques to the BP Model Repository technology.

The contribution of the thesis is threefold. Firstly, it contributes to the development of BP Model Repositories by providing the state of the art analysis. A framework for business process model repositories, including a management model and a reference architecture, is proposed to summarize the functionality of business process model repositories (Deliverable 1). A survey of business process model
repositories is provided to identify the functionality that need to be enhanced (Deliverable 2). Secondly, it contributes to the efficiency of process retrieval. After studying the state of the art of the process retrieval techniques (Deliverable 3), a technique is proposed to enhance the efficiency of process retrieval (Deliverable 4). Efficient process retrieval techniques are proposed, implemented and evaluated (Deliverable 5, 6, and 7). Thirdly, it contributes to the state of the art of BP Model Repository technology by integrating the proposed process retrieval techniques with APROMORE [53], one of the existing BP Model Repositories (Deliverable 8).

1.5 Thesis Structure

The thesis consists of four parts. Part I provides an introduction and an overview of this thesis. Part II analyzes functionality of BP Model Repositories and identifies that none of existing BP Model Repositories provides a technique to make process retrieval efficient. Part III proposes the techniques to enhance the efficiency of process retrieval. Part IV concludes the thesis. Figure 1.1 presents an overview of the remainder of this thesis, which is organized as follows.

Part II consists of two chapters. Chapter 2 provides a framework for BP Model Repositories (Deliverable 1). The framework consists of a management model and a reference architecture. The management model lists functionality that should be supported by BP Model Repositories. The reference architecture organizes the functionality into different layers and components. Chapter 3 provides a comparison between existing BP Model Repositories based on the framework proposed in Chapter 2 (Deliverable 2). In the comparison results we can see that efficient process retrieval technique is not supported by the existing BP Model Repositories (Deliverable 2).

Therefore, the goal of Part III is to enhance the efficiency of process retrieval. Part III consists of five chapters. Chapter 4 provides an overview of existing process retrieval techniques and retrieval techniques in related areas, e.g., the information retrieval and database areas (Deliverable 3). Chapter 5 describes a technique enhancing the efficiency of process retrieval (Deliverable 4). Chapter 6 presents the integration of the proposed technique with process retrieval techniques (Deliverable 5).
Chapter 1. Introduction

Figure 1.1: An overview of this Thesis

5). Chapter 7 presents the implementation of the proposed process retrieval techniques (Deliverable 6). Chapter 8 presents the experimental results (Deliverable 7).

Part IV consists of one chapter, which concludes the thesis by summarizing contributions and discussing limitations and future work (Deliverable 8).
Part II

Business Process Model

Repository
Chapter 2

A Framework for Business Process Model Repositories

2.1 Introduction

As described in Chapter 1, BP Model Repositories are required to manage large collections of business process models in organizations. However, a framework, summarizing the functionality that should be provided by BP Model Repositories, does not yet exist. Therefore, to provide an overview of the functionality of BP Model Repositories, this chapter analyzes and extends existing related work and provides a framework for BP Model Repositories. The contribution of this chapter is that it presents a descriptive framework for BP Model repositories, which is summarized by observing existing BP Model Repositories. The framework consists of a management model and a reference architecture. The management model lists the functionality that can be provided by BP Model Repositories, while distinguishing between functionality that is provided by general repositories and database management systems and functionality that is specific for repositories that contain business process models. The reference architecture presents the components that provide this functionality and their interconnections. The framework serves as a guide for the development of BP Model Repositories.

The remainder of this chapter is organized as follows. Section 2.2 presents the methodology of the chapter. Section 2.3 and Section 2.4 present the result
Chapter 2. A Framework for Business Process Model Repositories

of the framework development step, presenting a general BP Model Repository management model, which lists the functionality that can be provided by BP Model Repositories and presenting a reference architecture for BP Model Repositories that structures the functionality that can be provided. Section 2.5 shows the related work.

2.2 Approach

This section presents the approach of the chapter, consisting of literature study and framework definition.

Firstly, we have studied literature from the area of BP Model Repositories, i.e., the literature study step. In particular, we studied two types of papers in this area. First, we have searched and selected seminal work from the area of model repositories in general. Second, we have searched and selected existing BP Model Repositories. We followed a formal protocol to search and identity relevant literature [51]. More details about the strategy that we followed and the results that intermediate results that this lead to are provided in Appendix A.1. We have searched and selected seminal work from the area of model repositories in general, using secondary literature sources. We have selected the most frequently referenced papers as seminal work. In this way we have selected two seminal papers that provide an overview of the functionality that should be provided by repositories as well as an architecture. We have searched and selected work on existing BP Model Repositories, using secondary literature sources. The result is a collection of twenty BP Model Repositories that are briefly described in Appendix A.3.

Secondly, we have proposed a framework for BP Model Repositories, i.e., the framework definition step. The framework consists of a management model and a reference architecture.

To generalize a management model for BP Model Repositories, we perform the following two steps. First, we take all the functionality that a general repository provides [19], add them to the management model and divide them into three components [77]. The result of this step, an intermediate management model, is presented in Appendix A.2. Second, we observe how existing BP Model Repositories implement the functionality, modify the model based on what we observe, and record which functionality is modified based on the functionality of which BP Model
2.3 BP Model Repository Management Model

Repositories. The result of this step, the final management model, is described in Section 2.3.

To generalize a reference architecture for BP Model Repositories, we perform the following three steps. We start with a three-layer architecture consisting of a presentation, a logical, and a storage layer, because most of the existing BP Model repositories use a layered architecture with similar layers. Second, we divide the logical layer into the DBMS layer and repository management layer [19]. The result of this step, an intermediate reference architecture, is described in Appendix A.2. Third, we add process specific functionality into the repository management layer and re-label ‘process repository management layer’ to reflect these additions. Also, we add ‘external tools’ as a separate component, because external tools are also prominent in existing BP Model Repositories as tools that these repositories interact with. The result of this step, the final reference architecture, is in Section 2.4.

2.3 BP Model Repository Management Model

This section presents the functionality that BP Model Repositories can provide in the form of a management model. We consider a BP Model Repository as a specialized repository. The functionality for general repositories, as it is summarized by Bernstein and Dayal [19] and by Sagawa [77], can be specialized and extended to develop repositories that are specific for storing and managing business process models. We developed such an extension by taking the work of Bernstein and Dayal [19] and Sagawa [77] as a starting point and specializing and extending it, based on functionality that can be observed in BP Model Repositories and ongoing research, as we found it in the literature study. We cite the work from which each specialized or extended function is derived.

The BP Model Repository management model is shown in Table 2.1. It consists of three parts [77]: the process data model, the process function model and the process management model. Each model contains sub-models, which are further divided into elements.

2.3.1 Process Data Model

The process data model prescribes the kind of business process models and related data that can be stored in a BP Model Repository. It consists of the meta-model,
Chapter 2. A Framework for Business Process Model Repositories

The meta-model prescribes what information can be stored in the BP Model Repository by defining the concepts that are used in the repository and the relations between those concepts. Each BP Model Repository potentially supports a large number of concepts. We classify those concepts by identifying the process aspects and the types of process artifacts that are supported by a BP Model Repository. We distinguish the following process aspects.

- The activity aspect (A) contains concepts to describe the activities that are performed in the context of a process, e.g., the MIT process handbook and the process reuse architecture support it [41, 62].
- The control flow aspect (CF) contains concepts to describe the control flow relations between activities; these relations constrain the possible orders and combinations in which activities from the activity aspect can occur, e.g., BP-Suite and IPM support it [17, 25].
- The data aspect (D) contains concepts to describe the information that is used and changed during the execution of a process, e.g., BP-Suite and IPM support it [17, 25].
- The resource aspect (R) contains concepts to describe physical resources that are required to execute (activities in) a process, including human resources, e.g., BP-Suite and IPM support it [17, 25].
- The authorization aspect (Au) contains concepts to describe who is authorized to perform which part of a process, e.g., BP-Suite and IPM support it [17, 25].
- The organization aspect (O) contains concepts to describe the organizational structure, as it consists of people and organizational units, related to a collection of processes. The organizational structure is separate from the process model, but its elements may also appear in the resource or authorization aspects, e.g., BP-Suite and IPM support it [17, 25].
- The goal aspect (G) contains concepts to describe the hierarchy of strategic goals and to describe the relations of those goals to the processes that are meant to achieve them, e.g., the process reuse architecture [41].
2.3. **BP Model Repository Management Model**

<table>
<thead>
<tr>
<th>Process data model</th>
<th>Process meta model</th>
<th>Process aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Type of process artifacts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process presentation</th>
<th>Process notation</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Process storage model</th>
<th>External process data model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal process data model</td>
<td></td>
</tr>
<tr>
<td>Process related data model</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process index model</th>
<th>Process classifications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Other process indices</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process function model</th>
<th>Storage functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Create</td>
</tr>
<tr>
<td></td>
<td>Delete</td>
</tr>
<tr>
<td></td>
<td>Update</td>
</tr>
<tr>
<td></td>
<td>Import</td>
</tr>
<tr>
<td></td>
<td>Export</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Retrieval functions</th>
<th>Navigate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Search</td>
</tr>
<tr>
<td></td>
<td>Query</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integration functions</th>
<th>[Depend on external tools]</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Process management model</th>
<th>Process specific management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version management</td>
<td>Configuration management</td>
</tr>
<tr>
<td>Lifecycle management</td>
<td>Process view management</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General repository management</th>
<th>Access management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrity management</td>
<td>Transaction management</td>
</tr>
<tr>
<td>Checkin/out management</td>
<td>Dispatch management</td>
</tr>
<tr>
<td>Notification management</td>
<td>Notification management</td>
</tr>
<tr>
<td>Context management</td>
<td>Context management</td>
</tr>
</tbody>
</table>

- The relationship aspect (RS) contains concepts to describe the relationships between the process and other processes. These concepts can, for example, be used to represent a hierarchical structure of sub-processes and super-processes and to represent relations between process versions, e.g., IPM supports it [25]:
the relationship can also be generalization or specialization, e.g., the MIT process handbook supports it [62].

- The semantic aspect (S) contains concepts to annotate process semantically, e.g., the semantic business process repository supports it [60].
- The monitoring aspect (M) contains concepts to define how the performance of a process should be monitored, e.g., BP-Suite and IPM support it [17, 25].
- The management control (MC) aspect contains concepts to define the management controls that are implemented by (parts of) processes, e.g., the BPMN repository architecture supports it [85].

We distinguish the following types of process artifacts.

- A company specific process (C) is a process that is designed by a specific company to describe its own operations, e.g., BP-Suite and IPM support it [17, 25].
- A reference process (Re) is an abstract and standard process that can be reused and adapted to develop company specific processes. If a reference process contains pre-defined configuration options, it is also called a configurable reference process, e.g., the MIT process handbook and the process reuse architecture support it [41, 62].
- A process pattern (P) is a part of a process that describes a best practice, summarized from former experience, e.g., the process reuse architecture [41] supports it.
- A process instance (I), or case, is an execution of a process for a customer (a running instance of a process). It consists of some identifying information about the case (such as the customer’s name) and the activities that currently are or can be performed, e.g., BP-Suite supports it [17].
- Historical information (H) consists of logs that contain information about executions of the process instances. Typically it consists of information about the cases that were performed, the activities performed for these cases, when they were performed, who performed them and what data was processed, e.g., BP-Suite supports it [17].

The presentation dimension prescribes how information that is stored in a BP Model Repository is presented to the end-user, by associating a notation with its
concepts. For example, a BP Model Repository can store the information for activities and control flow relations between those activities, but that information can be presented to the user in (structured) natural language, in a standardized graphical notation like EPC or BPMN, or in a proprietary graphical notation. It is also common for a BP Model Repository not to prescribe a notation, but focus solely on defining its conceptual model and/or interchange format and leave the presentation of the business process models to external tools.

The storage model prescribes how the original information about the process must be technically provided to the BP Model Repository (external data model) and how it must be internally stored by the BP Model Repository (internal data model). The external and the internal model can be the same, for example each process can be stored as an XML file that is also used to exchange the process between the BP Model Repository and related tools, or they can differ, for example processes can be exchanged using XML but stored in a relational database, e.g., the process reuse architecture [41]. Other than that process related data, which is data that is used by, but not part of, the processes can be stored in the repository. Process related data includes: descriptors of web services that are used by the processes (e.g., BP-Suite stores WSDL files), and ontologies that are used to relate terms from different processes (e.g., the BPMN repository architecture [85] stores ontology information).

The index model prescribes the indices that are kept for process models, to allow both the user and the repository manager itself to quickly browse or search the collection of processes. An index that is commonly used is a classification of process models in terms of the business functions for which they are available. For example, we can classify processes into processes for: sales, procurement, production, finance and support. Subsequently, we can distinguish different classes of procurement processes, like procurement of product related materials and procurement of non-product related materials, etceteras. The MIT process handbook [62] and IPM [25] provide classifications.

2.3.2 Process Function Model

A BP Model Repository can support a series of basic functions to effectively manipulate the processes that it stores. We identify storage functions, retrieval functions and integration functions.
The storage functions are the functions to create (C), update (U) and delete (D) processes or parts of processes, by creating, updating or deleting instances of the concepts that are defined in the process meta model. In addition to that functions exist to import (I) complete processes into the repository, using the interchange format from the external data model, and to export (E) complete processes from the repository using that interchange format. For example, IPM [25] supports all five storage functions.

The retrieval functions can be used to obtain the required process according to some criteria. There are three methods for retrieving processes: navigate, query and (similarity) search. Navigation is the method of manually scanning processes in a list, or by using a classification or some other index. Search provides the function to get similar processes with respect to the given query, which can be a (fragment of a) process or keywords. Query provides more advanced functions to specify search criteria using a query language, such as IPM-PQL [25] or BPMN-Q [6]. Retrieval functions can have a focus on one or more process aspects or types of artifacts. Awad distinguishes the following three focuses: retrieving (elements of) process definitions (company specific or reference), retrieving (elements of) process instances and retrieving (elements of) process execution history [6].

The integration functions can be used to integrate a process repository with external tools. Integration varies, depending on the types of tools a repository integrates. In the BP Model Repositories that we have studied, we have observed integrations with the following types of tools.

- Process modeling tools, which can be used to visually create, retrieve, update and delete processes, e.g., IPM [25] provides it.
- Report generator, which can be used to generate reports about (monitoring information of) processes and their properties, e.g., BP-Suite [17] provides it.
- Process analysis tools, which can be used to analyze correctness, selected properties or performance of processes, e.g., IPM [25] provides it.
- Process engines, which can be used to execute business processes by performing activities, or notifying human resources that activities must be performed, according to the order specified by the control flow relations. When executing a business process a process instance is created and monitoring information is generated, e.g., IPM [25] and BP-Suite [17] provide it.
• Process administration and monitoring tools, which can be used to manage or monitor the (executions of) processes, e.g., BP-Suite [17] provides it.

• Collaboration tools, which can be used to establish business collaborations based on processes in the repository, e.g., the BPMN repository architecture [85] provides it.

Within the set of BP Model Repositories that we studied, there was no strict separation with respect to what is considered internal functionality of the repository and what is considered external functionality that can be integrated with the repository. For example, query tools have been proposed as external tools [6], but at the same time tools for establishing collaborations between organizations, based on their processes, have been proposed as internal parts of the repository [85]. We made the separation between internal and external functionality above, based on what we most frequently observed in the analyzed BP Model Repositories.

2.3.3 Process Management Model

Advanced management functions can be subdivided into functions that are provided by general repositories and functions that are provided only by BP Model Repositories.

The process specific management functions are: version management [60], configuration management [25], lifecycle management [25] and view management [80]. Although version management, configuration management and view management are also general repository functions (or even general database functions in the case of view management), these functions have been specialized to meet process specific requirements [1, 25, 29, 54, 105]. The version management function enables multiple versions of the same process or activity to be maintained simultaneously, as well as differences between these versions to be analyzed and similarities to be traced. The configuration management function makes it possible to store and maintain the relation between (a version of) a process and the (versions of) subprocesses and activities that it consists of. Although version and configuration management are also general repository functions, specialized functionality is added to support requirements in the context of BP Model Repositories. For example, when a process is being executed and a new version of that process or a part of that process is created, a decision must be made as to whether the new version will be put into
Chapter 2. A Framework for Business Process Model Repositories

effect for process instances that are already running or not and, if so, for which process instances. The lifecycle management function keeps track of the stage in its lifecycle that a process is currently in. For example, a process can be under design, validation and current. Depending on the stage that it is in, the lifecycle manager ensures that some operations can be performed on the process while others cannot. For example, a new version cannot be created of a process that is still in the design phase of the lifecycle, nor can a process that is still in validation phase of the lifecycle be executed. The view management function makes it possible to create multiple views on the same process. Although view management is a general database function, specialized functionality is added to support requirements in the context of BP Model Repositories. For example, it is common to keep a private view on a process, which represents the process as it is performed inside an organization. At the same time a public view (also called service) can be provided of what the behavior of the process to the outside world will be like, therewith preserving company secrets of how services are internally implemented and not bothering clients with details that do not concern them. To support the generation of the public view from the private view and to keep the two views consistent, BP Model Repository specific functionality is needed.

The general repository management functions are: access management, integrity management, transaction management, checkin/out management, dispatch management, notification management and context management. The access management function ensures that people only have access to the objects in the repository that they are authorized to view. The integrity management function ensures that the data in the repository is accurate, valid, and consistent. Transaction management ensures that multiple operations on a repository can be performed in a transactional manner (i.e.: either all at once or not at all). Checkin/out management allows a user to check-out objects from the repository, therewith locking them so others cannot change them, make the desired changes and then check them in again by releasing the lock. Optionally, multiple people can be allowed to check-out an object at the same time, in which case check-in management should ensure that changes that are made to the same object by multiple people are properly merged. Dispatch management makes it possible to associate a work-order with an object, such that it is forwarded to people in the order specified in the work-order along with notes about what these people have to do with the object. (This is usually
called ‘workflow management’ in repositories [19]. We call it dispatch management to avoid the confusion with workflow tools that execute the processes in a BP Model Repository. Notification management enables notifications to be generated in case an object in the repository is changed. The context management function allows collections of repository objects, called ‘contexts’, (also called ‘projects’ or ‘workspaces’) to be created and manipulated. Contexts can be stored persistently.

2.4 BP Model Repository Reference Architecture

This section presents a reference architecture for BP Model Repositories. The reference architecture is developed by investigating the architectures of the existing BP Model Repositories as introduced in Section 2.2. In that respect it should be characterized as a descriptive reference architecture (describing the commonalities in the architectures of existing BP Model Repositories), rather than a prescriptive reference architecture (prescribing how a BP Model Repository architecture should be structured) [5].

Figure 2.1 shows the reference architecture. It consists of two part: the BP Model Repository and external tools. External tools provides integration functions for BP Model Repositories as described in the previous section.

The architecture of BP Model Repositories has four layers: the presentation layer, the process repository management layer, the database management layer and the storage layer.

The presentation layer provides the UI for users to interact with the BP Model Repository, so the users can easily interact with the functions provided by the repository.

The process repository management layer provides both process specific repository functions and functions that are provided by general repositories. The functions are described in detail in the previous section. Although general database management systems implement general functions, such as querying, most BP Model Repositories implement these functions themselves, because this allows them to at least provide a facade that applies the functions specifically to processes instead of database tables.

The database management layer provides the functions that are generally provided by database management systems, and the functions are described in the
previous section.

The storage layer manages the storage of the process models, the related data and indices or classifications to enable fast querying, searching and navigation of the BP Model Repository. Process models can be stored both in an internal format, for example as rows in database tables, and in their original external format. In that
In most cases the storage layer is implemented by a general database management system. Relational (e.g., the MIT process handbook [62]), object-oriented (e.g., the BPMN repository architecture [85]) and XML databases (e.g., IPM [25]) have all been observed in the concrete BP Model Repositories that were studied. Alternative implementations that have been observed are implementations using general repositories, of which one using a distributed repository, i.e., OSIRIS [79], and an implementation in which the data is stored as files in a filesystem, i.e., the library for process programming [104].

Well-defined interfaces should exist between the different layers. In most BP Model Repositories well-defined interfaces exist between the presentation layer and the process repository management layer and the database management layer. The technology that is used to implement the interfaces varies. The process repository interface can be implemented using a programming language API, but also using remote method invocation or even using web-services. The DBMS interface can be implemented using a (standard) API, but we have also observed concrete BP Model Repositories that added an additional layer that abstracts from the storage technology that was used, to allow different storage technology to be used without having to implement the repository functions.

In addition to the interfaces between the layers, interfaces can exist between the BP Model Repository and external tools. The presence and implementation of these interfaces varies largely. However they are all defined either to interact with the presentation layer, with the process repository management layer or with both. Interaction with the presentation layer enables the BP Model Repository to open a tool from the UI of the BP Model Repository. Interaction with the process repository management layer enables an external tool to directly invoke the functions that are provided by the BP Model Repository.

The functionality listed in the management model are included in the reference architecture. The process data model is implemented in the storage layer; the process function model is implemented in the DBMS layer, the process repository management layer, and the external tools; while the process management model is implemented in the DBMS layer and the process repository management layer.
Chapter 2. A Framework for Business Process Model Repositories

2.5 Related Work

The criteria and requirements for BP Model Repositories are presented in [81, 82]. The framework in this thesis differs from the criteria and requirements that are presented in [81, 82] in two ways. Firstly, the framework in this chapter is descriptive describing what functionality a BP Model Repository can support, while the criteria and requirements are prescriptive prescribing what functionality a BP Model Repository should support. Secondly, the framework in this chapter focuses on functional requirements for BP Model Repositories, while the scope of the criteria and requirements proposed in [81, 82] is more broad, also covering interface criteria and non-functional requirements. As a consequence, the criteria and requirements proposed in [81, 82] are more detailed with respect to interface criteria and non-functional requirements, while this chapter is more detailed with respect to provided functionality.
Chapter 3

A Comparison of Business Process Model Repositories

3.1 Introduction

In Chapter 2 a framework for BP Model Repositories is proposed. This chapter uses the framework to compare the state of the art in BP Model Repositories. To this end, it analyzes to what extent the functionality, that is presented by the framework, is supported by each of the twenty BP Model repositories that were identified during literature study. We followed a formal protocol to search and identify relevant literature [51]. More details about the strategy that we followed and the results that intermediate results that this lead to are provided in Appendix A.1 and an overview of these repositories are provided in Appendix A.3. As a result, we can observe which functionality for BP Model Repositories is well supported and which is not.

The comparison is done based on literature. We consider a function to be supported by a repository, if the papers that describe the repository claim that that function is supported.

The remainder of this chapter is organized as follows. Section 3.2, 3.3, and 3.4 presents the comparison of existing BP Model Repositories based on process data, function, and management model respectively, which are three parts of the framework. Section 3.5 provides statistics about the comparison.
Chapter 3. A Comparison of Business Process Model Repositories

3.2 Process Data Model Comparison

Table 3.1 shows the comparison of how existing BP Model Repositories implement the process data model. The aspects and model types that are supported by a BP Model Repository are identified by one or two letters as they are defined in section 2.3 and summarized below the table.

With respect to support for the process aspect, the table shows that, except for the MIT Process Handbook, all existing BP Model Repositories consider storage of the control flow aspect. The MIT Process Handbook focuses on storing textual descriptions of the processes and the activities that occur within those processes; it shows the order in which activities usually occur in a process, but no comprehensive control flow. In addition to control flow, the data and authorization aspect are frequently supported. There is a strong relation between whether or not the resource aspect is supported and whether or not an existing BP Model Repository integrates with a process engine; out of the 7 BP Model Repositories that integrate with a process engine, only the Process Library does not support modeling the resource aspect. The relation between resource aspect support and engine integration is expected, because a process engine has to distribute work to resources. The Process Library relies on external tools to perform the distribution to resources; it focuses solely on the control flow and data aspects. The management control aspect is only supported by the Process Reuse Architecture and the BPMN Repository Architecture. Both these BP Model Repositories also support the monitoring aspect and enable their users to relate monitoring information to controls, to relate values for performance indicators to controls.

With respect to support for the types of artifacts, most tools focus on storing company-specific business process models. Two classes can be recognized for the BP Model Repositories that also support reference processes. The first class is the class of BP Model Repositories that focus on storing reference processes to make them available as a knowledge base. The MIT Process Handbook and IPR fall into this class. The second class is the class of BP Model Repositories that focus on adapting reference processes to develop company-specific processes. The Process Reuse Architecture, the BPMN Repository, ProcessGene, the Querying Framework and Prosero fall into this class.
<table>
<thead>
<tr>
<th>aspect*</th>
<th>type*</th>
<th>notation</th>
<th>external</th>
<th>internal</th>
<th>related</th>
<th>classifications</th>
<th>others</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIT Pro-cess Handbook</td>
<td>A, RS</td>
<td>Re</td>
<td>Structured natural language</td>
<td>Database</td>
<td>Part-Whole, Generalization-Specialization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Reuse Architecture</td>
<td>A, CF, D, R, Au, O, G, MC, M</td>
<td>Re, C, P</td>
<td>Structured natural language</td>
<td>XML</td>
<td>Database</td>
<td>Facets, Patterns</td>
<td></td>
</tr>
<tr>
<td>Process library</td>
<td>A, CF, D, RS</td>
<td>C</td>
<td>P language</td>
<td>P language</td>
<td>Part-Whole, Generalization-Specialization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPM</td>
<td>A, CF, D, R, Au, O, RS, M</td>
<td>C</td>
<td>Proprietary (graphical)</td>
<td>IPM-EPDL</td>
<td>Repository objects</td>
<td>Categories</td>
<td></td>
</tr>
<tr>
<td>RepoX</td>
<td>A, CF, D, RS</td>
<td>C</td>
<td>Not specified</td>
<td>XML</td>
<td>Database (tables/objects)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workflow repository</td>
<td>A, CF, D, R, RS, M</td>
<td>C, I</td>
<td>Not specified</td>
<td></td>
<td>Database</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meta Presentation Storage Index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>aspect</strong></td>
<td><strong>type</strong></td>
<td><strong>notation</strong></td>
<td><strong>external</strong></td>
<td><strong>internal</strong></td>
<td><strong>related</strong></td>
<td><strong>classifications</strong></td>
<td><strong>others</strong></td>
</tr>
<tr>
<td><strong>Oryx</strong></td>
<td>A, CF</td>
<td>C</td>
<td>BPMN, EPC, Petri nets</td>
<td>RDF</td>
<td>Database</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BP-Suite</strong></td>
<td>A, CF, D, R, Au, O, M</td>
<td>C, I, H</td>
<td>BPEL</td>
<td>BPEL</td>
<td>Database</td>
<td>WSDL</td>
<td></td>
</tr>
<tr>
<td><strong>Process-</strong></td>
<td>A, CF, Au</td>
<td>Re, C</td>
<td>Not specified</td>
<td>Database</td>
<td>Categories</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Gene</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PVR</strong></td>
<td>A, CF, D, R, Au, M</td>
<td>C, I, H</td>
<td>Proprietary (graphical)</td>
<td>Database</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Querying</strong></td>
<td>A, CF, D, G, R, Au, S</td>
<td>Re, C, P</td>
<td>Not specified</td>
<td>WSML</td>
<td>Repository Objects (RDF-like)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Framework</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SBPR</strong></td>
<td>A, CF, D, Au, R, RS, S</td>
<td>C</td>
<td>BPMO, sBPMN, sEPC, sBPEL</td>
<td>WSML</td>
<td>Database (tables)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IBM BPEL</strong></td>
<td>A, CF, D, Au</td>
<td>C</td>
<td>BPEL</td>
<td>BPEL</td>
<td>Repository Objects (EMF)</td>
<td>WSDL</td>
<td></td>
</tr>
<tr>
<td><strong>Repository</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Prosero</strong></td>
<td>A, CF, D, R, Au, O, S</td>
<td>Re, C, I</td>
<td>BPMN, BPEL</td>
<td>XML</td>
<td>Database</td>
<td>Terminology, WSDL</td>
<td></td>
</tr>
</tbody>
</table>
### 3.2. Process Data Model Comparison

<table>
<thead>
<tr>
<th>aspect*</th>
<th>type*</th>
<th>notation</th>
<th>external</th>
<th>internal</th>
<th>related</th>
<th>classifications</th>
<th>others</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSIRIS</td>
<td>A, CF, D</td>
<td>C, I</td>
<td>Proprietary</td>
<td>Repository</td>
<td>WSDL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APRO-MORE</td>
<td>A, CF, D, R</td>
<td>C, Re, P</td>
<td>Proprietary</td>
<td>XML</td>
<td>Database</td>
<td>Categories</td>
<td></td>
</tr>
<tr>
<td>Fragmento</td>
<td>A, CF, D, Au</td>
<td>C, P</td>
<td>BPEL</td>
<td>BPEL</td>
<td>Database (Object-relational)</td>
<td>WSDL</td>
<td></td>
</tr>
<tr>
<td>BPCM</td>
<td>A, CF, R, Au, G, S</td>
<td>C</td>
<td>BPCM meta-model</td>
<td>XML</td>
<td>Database (Rational)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPR</td>
<td>A, CF</td>
<td>C, Re</td>
<td>Not specified</td>
<td>Not specified</td>
<td>Repository (Objects)</td>
<td>Categories</td>
<td></td>
</tr>
</tbody>
</table>

* Activity(A), Control Flow(CF), Data(D), Resource(R), Authorization(Au), Organization(O), Goal(G), Relationship(RS), Semantic(S), Monitoring(M), and Management Control(MC). (See Section 2.3.1)

* Company specific(C), Reference process(Re), process Pattern(P), process Instance(I), and Historical information(H). (See Section 2.3.1)
With respect to support for the \textit{process notation}, BPEL and BPMN are the most common notations. BPEL is supported by five BP Model Repositories, BP-Suite, SBPR (it supports sBPEL), IBM BPEL repository, Prosero, and Fragmento. BPMN is supported by four Repositories, the BPMN repository, Oryx, SBPR (it supports sBPMN), and Prosero.

With respect to support for the \textit{external process data model}, most repositories either use XML or an XML-based standard (IPM-EPDL, BPEL, WSML, BPMO, sBPMN, sEPC or sBPEL). The exceptions are the MIT Process Handbook and the Process Library. The MIT Process Handbook is available only through a web-based user interface, through which users interact using natural language. The Process Library stores and exchanges its models through the P-Language, which strongly resembles a programming language.

With respect to support for the \textit{internal process data model}, most BP Model Repositories store processes in a database. They either store them in their external format as ‘blobs’ or ‘clobs’ in a database or they store them in a more fine-grained manner, keeping separate tables or objects for separate elements (for example using a separate table for tasks, for processes and for data elements). Six BP Model Repositories, IPM, IPR, OSIRIS, the BPMN Repository, the Querying Framework and the IBM BPEL Repository, use a general repository instead of a database. In the repository the processes are stored as repository objects. The process library stores processes as files in the file system.

With respect to support for the \textit{process related data model}, a few BP Model Repositories store process-related information. This is limited to storing ontologies that establish a unified terminology and web service descriptors that define the web-services that are invoked by the processes in the repository.

With respect to support for the \textit{process classifications}, some BP Model Repositories allow their users to flexibly define process categories and classify the processes in these categories. Pre-defined classifications that have been proposed are a classification according to part-whole relations, a classification according to generalization-specialization relations, a classification of process patterns and a classification of process facets. Seven out of the total of twenty BP Model Repositories use a classification scheme.

With respect to support for the \textit{other process indices}, none of the BP Model Repositories provides it.
3.3. Process Function Model Comparison

Table 3.2 shows the comparison of how existing BP Model Repositories implement the process function model.

<table>
<thead>
<tr>
<th>Process function model</th>
<th>Storage*</th>
<th>Retrieval</th>
<th>Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Handbook</td>
<td>Restricted</td>
<td>Search, Navigation of Definition</td>
<td></td>
</tr>
<tr>
<td>Reuse Architecture</td>
<td>C,D,U,I,E</td>
<td>Search, Navigation of Definition</td>
<td></td>
</tr>
<tr>
<td>Process library</td>
<td>File system</td>
<td>Search, Navigation of Definition</td>
<td>Engine</td>
</tr>
<tr>
<td>IPM</td>
<td>C,D,U,I,E</td>
<td>Query, Navigation of Definition (IPM-PQL)</td>
<td>Modeling, Engine, Analysis</td>
</tr>
<tr>
<td>RepoX</td>
<td>C,D,U,I,E</td>
<td>Query, Navigation of Definition (SQL, XQuery)</td>
<td></td>
</tr>
<tr>
<td>Workflow Repository</td>
<td>C,D,U</td>
<td>Query of Definition, Instances (SQL)</td>
<td>Engine, Analysis</td>
</tr>
<tr>
<td>BPMN Repository</td>
<td>C,D,U,I,E</td>
<td></td>
<td>Engine, Modeling, Collaboration</td>
</tr>
<tr>
<td>Oryx</td>
<td>C,D,U,I,E</td>
<td>Query of Definition (BPMN-Q)</td>
<td>Modeling</td>
</tr>
<tr>
<td>ProcessGene</td>
<td>C,D,U</td>
<td>Query of Definition (SQL)</td>
<td></td>
</tr>
<tr>
<td>PVR</td>
<td>C,D,U</td>
<td>Query of Definition, Instances (Proprietary)</td>
<td>Engine, Analysis</td>
</tr>
<tr>
<td>Querying Framework</td>
<td>C,D,U,I,E</td>
<td>Query, Navigation of Definition (WSML)</td>
<td>Modeling</td>
</tr>
</tbody>
</table>
### Table 3.2 – Continued

<table>
<thead>
<tr>
<th>Process function model</th>
<th>Storage*</th>
<th>Retrieval</th>
<th>Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBPR</td>
<td>C,D,U,I,E</td>
<td>Query of Definition (WSML)</td>
<td>Engine</td>
</tr>
<tr>
<td>BPEL Repository</td>
<td>C,D,U,I,E</td>
<td>Query of Definition (OCL)</td>
<td>Engine(OCL)</td>
</tr>
<tr>
<td>Prosero</td>
<td>C,D,U,I,E</td>
<td>Modeling, Engine, Analysis</td>
<td></td>
</tr>
<tr>
<td>OSIRIS</td>
<td>C,D,U</td>
<td>Modeling</td>
<td></td>
</tr>
<tr>
<td>APROMORE</td>
<td>C,D,U,I,E</td>
<td>Search, Navigation of Definition</td>
<td></td>
</tr>
<tr>
<td>Fragmento</td>
<td>C,D,U,I,E</td>
<td>Search, Query of Definition (SQL)</td>
<td>Modeling</td>
</tr>
<tr>
<td>BPCM</td>
<td>C,D,U,I,E</td>
<td>Query, Navigation of Definition (SQL)</td>
<td>Modeling</td>
</tr>
<tr>
<td>IPR</td>
<td>C,D,U</td>
<td>Search, Navigation of Definition</td>
<td>Modeling, Analysis</td>
</tr>
</tbody>
</table>

* Create(C), Update(U), Delete(D), Import(I), and Export(E). (See Section 2.3.2)

With respect to support for the process storage functions, most BP Model Repositories support the create, update and delete storage functions. Exceptions are the MIT Process Handbook and the Process Library. Although, strictly speaking both BP Model Repositories do allow processes to be created, updated or deleted, they do not provide a public interface to do that. The MIT Process Handbook only allows for maintenance by certain people and the Process Library uses the file system for storage, but does not provide repository functionality for that purpose. We say that a BP Model Repository supports import and export of models if and only if they have an external file format. Files can then be imported or imported in that external file format.

With respect to support for the process retrieval functions, most BP Model Repositories support one or more functions to search, query or navigate the repos-
itory. Reading the papers in which the repositories are described leads to the conclusion that providing such functionality is an important motivation for developing a process repository. The support for such functionality is diverse. The table shows which type of functionality each BP Model Repository supports: search, query or navigation. It also shows for which types of process each BP Model Repository supports retrieval functionality. Retrieval functionality can be supported for process definitions, which includes definitions of reference processes, company-specific processes and process patterns, depending on the types of processes that the BP Model Repository supports. Retrieval functionality can also be supported for process instances and for historical information about process instances that have been running in the past. If applicable, the table shows the query language that is used by the repository. Interestingly, only IPM, BPMN-Q, BP-Suite and PVR provide a process specific query language. The other BP Model Repositories use a general purpose query language. The Querying Framework supports a graphical manner for specifying queries by selecting a part of a process model. Although Prosero, OSIRIS, and the BPMN Repository do not explicitly claim to support process search, they should be able to provide such functionality through the (SQL-based) search mechanisms that general databases or repositories provide.

With respect to support for the process integration functions, the level of integration of the BP Model Repositories with other tools is low, especially considering that BP Model Repositories are meant as supporting technology; they are only meant to store process models, while those process models must be designed in external tools and operations on them must also be performed by external tools. We attribute the lack of integration with external tools to the fact that most of the repositories are research prototypes, developed specifically to show the feasibility of and demonstrate the repository technology.

### 3.4 Process Management Model Comparison

Table 3.3 shows the comparison of how existing BP Model Repositories implement the process management model.

With respect to support for the general repository management, all BP Model Repositories use a database management system or a general repository to support storage of processes except for the Process Library. The table shows which type of
system is used. Theoretically, this means that the management functions that are provided by these systems are available to the users of the BP Model Repository. However, in practice, additional effort may be required to make these functions available in a practical manner. For example, to make access control available in a practical manner for a database in which different types of process elements are stored in different tables, a single user must be given access to each relevant table to be allowed to create or read a single process. To make access control more practical in such cases security roles should be defined that package the access to all tables relevant to a process.

With respect to support for the process specific management, the BP Model Repositories rarely support it. For example, only one out of twenty BP Model Repositories supports process lifecycle management.

<table>
<thead>
<tr>
<th>Table 3.3: The process management model comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process management model</td>
</tr>
<tr>
<td>General repository</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>Process Handbook</td>
</tr>
<tr>
<td>Reuse Architecture</td>
</tr>
<tr>
<td>Process library</td>
</tr>
<tr>
<td>IPM</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>RepoX</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Workflow repository</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>BPMN Repository</td>
</tr>
<tr>
<td>Oryx</td>
</tr>
<tr>
<td>BP-Suite</td>
</tr>
<tr>
<td>ProcessGene</td>
</tr>
<tr>
<td>PVR</td>
</tr>
<tr>
<td>Querying Framework</td>
</tr>
</tbody>
</table>
3.5 Ratio of Implemented Functionality

Based on the results of the comparison, Table 3.4 provides an overview of the ratio of BP Model Repository Functionality implemented by each BP Model Repository. The rows of the table contain the different BP Model Repositories and the columns contain the classes of functionality in the process management model from Table 2.1. The cells show how many of the functions in each class, each repository implements. In the table we can see that IPR implements most of the functions (twenty four out of twenty eight) and the process library implements the least (nine out of twenty eight).

**Table 3.4:** An Overview of Functionality Implementation Ratios of each BP Model Repository

<table>
<thead>
<tr>
<th>Process management model</th>
<th>Storage</th>
<th>Function</th>
<th>Management</th>
<th>Overall</th>
</tr>
</thead>
</table>
### Table 3.4 – Continued

<table>
<thead>
<tr>
<th>Process management model</th>
<th>Storage</th>
<th>Function</th>
<th>Management</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuse Architecture</td>
<td>6/8</td>
<td>7/9</td>
<td>3/11</td>
<td>16/28</td>
</tr>
<tr>
<td>Process library</td>
<td>6/8</td>
<td>3/9</td>
<td>0/11</td>
<td>9/28</td>
</tr>
<tr>
<td>IPM</td>
<td>6/8</td>
<td>8/9</td>
<td>10/11</td>
<td>24/28</td>
</tr>
<tr>
<td>RepoX</td>
<td>5/8</td>
<td>7/9</td>
<td>5/11</td>
<td>17/28</td>
</tr>
<tr>
<td>Workflow repository</td>
<td>4/8</td>
<td>5/9</td>
<td>5/11</td>
<td>14/28</td>
</tr>
<tr>
<td>BPMN Repository</td>
<td>6/8</td>
<td>6/9</td>
<td>7/11</td>
<td>19/28</td>
</tr>
<tr>
<td>Oryx</td>
<td>5/8</td>
<td>7/9</td>
<td>3/11</td>
<td>15/28</td>
</tr>
<tr>
<td>BP-Suite</td>
<td>6/8</td>
<td>7/9</td>
<td>3/11</td>
<td>16/28</td>
</tr>
<tr>
<td>PVR</td>
<td>4/8</td>
<td>5/9</td>
<td>3/11</td>
<td>12/28</td>
</tr>
<tr>
<td>Querying Framework</td>
<td>6/8</td>
<td>8/9</td>
<td>7/11</td>
<td>21/28</td>
</tr>
<tr>
<td>SBPR</td>
<td>5/8</td>
<td>7/9</td>
<td>4/11</td>
<td>16/28</td>
</tr>
<tr>
<td>BPEL Repository</td>
<td>6/8</td>
<td>7/9</td>
<td>7/11</td>
<td>20/28</td>
</tr>
<tr>
<td>Prosero</td>
<td>6/8</td>
<td>6/9</td>
<td>3/11</td>
<td>15/28</td>
</tr>
<tr>
<td>OSIRIS</td>
<td>5/8</td>
<td>4/9</td>
<td>7/11</td>
<td>16/28</td>
</tr>
<tr>
<td>APROMORE</td>
<td>6/8</td>
<td>7/9</td>
<td>3/11</td>
<td>16/28</td>
</tr>
<tr>
<td>Fragmento</td>
<td>6/8</td>
<td>8/9</td>
<td>5/11</td>
<td>19/28</td>
</tr>
<tr>
<td>BPCM</td>
<td>5/8</td>
<td>8/9</td>
<td>6/11</td>
<td>19/28</td>
</tr>
<tr>
<td>IPR</td>
<td>6/8</td>
<td>6/9</td>
<td>7/11</td>
<td>19/28</td>
</tr>
</tbody>
</table>
3.5. Ratio of Implemented Functionality

Figure 3.1 provides an overview of the ratio of implementation of the different functionality in the framework. The figure shows, for each function, how many out of twenty existing BP Model Repositories implement that function. In the figure we can see that all the BP Model Repositories store internal process models and provide a UI. These functions thus clearly constitute the basic functionality that is required by any BP Model Repository. The figure also shows that most of process specific functions are not frequently implemented. For example, only two out of twenty BP Model Repositories support process view management and only one out of twenty BP Model Repositories integrate process monitoring tool and report generator. The reason for that can be sought in the fact that this chapter studies research prototypes, which can be expected to implement a particular function with respect to which the research has been conducted and few other functions.
Figure 3.1: An Overview of Functionality Implementation Ratios in the Architecture
Part III

Efficient Process Model Retrieval
Chapter 4

Introduction of Process Retrieval

4.1 Introduction

Part II presents a framework for BP Model Repositories and compares existing ones based on the framework. The framework lists the functions that BP Model Repositories can provide and the survey shows to what extent existing BP Model Repositories implement these functions. In this part, we focus on enhancing one of the essential functions of BP Model Repositories, i.e., process retrieval. From the survey we know that seventeen out of twenty existing BP Model Repositories we have surveyed explicitly support process retrieval (Table 3.2) and the other three also support retrieval through general databases or repositories (Table 3.3). In these existing BP Model Repositories, retrieval techniques are divided into three branches: process specific retrieval techniques (BPMN-Q [6], BP-QL [14], IPM-PQL [25], and PVR [59]), XQuery based retrieval techniques and SQL based retrieval techniques. However, there is no indexing technique for process specific and XQuery based retrieval techniques other than classifications of process models (Table 3.1); while default indexing techniques in databases for SQL are designed for general information (data) retrieval instead of process retrieval, as such retrieving using SQL based retrieval techniques still take considerable time [10]. Therefore, in this part a process specific index that is designed for process retrieval techniques is proposed.
Chapter 4. Introduction of Process Retrieval

and applied to these techniques to make them more efficient.

As explained in Section 3.3, there are three retrieval functions, i.e., navigation, query, and (similarity) search; each retrieval function has three focuses, i.e., definition, instance, and trace. In this part, we consider process query and similarity search, because, for navigation, classifications of process models are sufficient. The focus of this part is process definition, because the goal of this part is providing an indexing technique that can be integrated with existing process retrieval techniques to enhance their efficiency and most of existing techniques focus on process definitions, as shown later in Table 4.1 and 4.2.

This chapter introduces process retrieval by providing examples and summarizing related work. The remainder of the chapter is organized as follows. Section 4.2 presents an example and recent work of process querying. Section 4.3 presents an example and recent work of process similarity search. Section 4.4 presents related work in other related research fields.

4.2 Process Querying

Querying a collection of business process models is done by retrieving all models in the collection that satisfy a given query. In this thesis, a query is described by a fragment of a process models (it can also be described by query languages [25]). The queries support both basic and advanced elements. Basic elements are notational elements of process modeling notations like BPMN, EPCs, and Petri nets. For example, Figure 4.1 shows the core elements of BPMN: flow objects, connecting objects, swimlanes, and artifacts\(^1\). Advanced elements are wildcard nodes, transitive edges, negative edges, and neg-transitive (negative transitive) edges [6, 14, 25]. A wildcard node matches any node in a business process graph. A transitive edge matches a sequence of edges starting from its source node to its target node. A negative edge matches if there is no edge between its source node and its target node. A neg-transitive edge is transitive and negative at the same time, matching if there is no sequences from its source node to its target node. For example, query a from Figure 4.2 is a query with only basic elements, and the first three models should be retrieved, because the query fragment is a sub-model of all of them. While query b from Fig. 4.2 contains a transitive edge between node ‘Order’ and ‘Receive’ and

\(^1\)http://www.omg.org/bpmn/Samples/Elements/Core_BPMN_Elements.htm
4.2. Process Querying

a negative edge between node ‘Receive’ and ‘Pay’; the first four models should be retrieved (detailed steps of process querying are given in Chapter 6).

![Figure 4.1: Core Set of BPMN Elements](image)

![Figure 4.2: Querying a Collection of Business Process Models](image)

To the best of our knowledge, there exists seven other techniques that focus on the issue of process querying. Table 4.1 compares these techniques and the technique in this thesis (listed in the rows) with respect to the following aspects (listed in the columns): focus, advanced, index, and evaluation. Focus refers to the focus of process querying, i.e., definition, execution, or trace. Advanced refers to whether a technique supports advanced query elements. Index refers to whether a technique provides a process specific index.

Three process query languages, focusing on querying process definitions, have been proposed [6, 14, 25]. Awad [6] develops BPMN-Q, a graphical language to query business processes modeled in BPMN, by extending the BPMN notation. BPMN-Q supports both basic and advanced queries. The basic queries it supports are process fragments models in BPMN; the advanced queries it supports contains
variants of BPMN elements, e.g., a node that can represent any gateway. Beeri et al. [14] propose BP-QL, a graphical language to query business processes modeled in BPEL. BPEL describes the process of a service, each process activity is supported by another service described in a WSDL file (an atom service providing the functionality) or another BPEL (a sub-process) file. As such, BP-QL supports queries that can match with processes after unfolding all sub-processes. Choi et al. [25] propose IPM-PQL, a query language for a proprietary process modeling notation (IPM-EPDL) based on XML. IPM-PQL supports process specific queries like searching processes that have a certain activity or a certain transition from one activity to another. Jin et al. [46] develop an indexing technique for process querying supporting basic queries. It takes sequences in process models and constructs an index using these sequences. All the work above is querying definitions (notations) of business processes, while there is also work that is querying executions and traces of business processes. Beeri et al. [15] propose BP-Mon, a graphical language to monitor executions of BPEL processes. It supports monitoring activities and associated reports, e.g., the execution time of an activity and the occurrences of an activity in one execution. Beheshti et al. [18] propose FPSPARQL, a query language to analyze business process execution traces. It models process traces as a graph (using folder nodes and path nodes) and then query the graph using FPSPARQL, an extension of SPARQL [73] (a W3C standard for querying RDF). Balan et al. [11] propose BP-Ex, a graphical query language to analyze business process execution traces. It supports analyzing properties of a certain pattern described in a query, e.g., OLAP
functions (drill down, roll up, etc.).

From Table 4.1 we can see that our technique in this thesis is the only one providing a process specific indexing technique for advanced process querying (columns Advanced and Index). Sequence based index [46] improves the efficiency for basic queries but not for advanced queries. Process query languages [6, 14, 25] work on the expressive power of the language, rather than on the efficiency of retrieving models that satisfy a given query. Consequently, performing an advanced business process query can take considerable time. For example, it on average takes 5s to run a query with a collection of 500 process models, using BPMN-Q [10]. While users of a search engine typically expect a response within milliseconds.

4.3 Process Similarity Search

Given a query model, a process similarity search technique should retrieve those process models that are similar to the query model and it should return those similar process models in order of their similarity to the query model. More precisely, for a query model and a model in a collection, the technique should provide functions that result in a similarity score between 0 and 1, where 0 indicates no similarity and 1 indicates identical models, and the models are ranked descendingly according to these scores. In this thesis, the ratio of shared features (fragments of process models) are used as the function to estimate similarity and graph edit distance are used as the function to perform final checking. Both query models and models in a collections are described by notational elements of process modeling notations, e.g., BPMN, EPCs.

Figure 4.3 shows an example of business process similarity search. It shows one search model and five process models in the BPMN notation. Suppose that we use nodes and sequences of size 1 as features of models in Fig. 4.3. We can observe that in model 1 there are six matching features with the query model: the task features ‘Buy Goods’, ‘Receive Goods’ and ‘Verify Invoice’; and the sequence features (‘Buy Goods’, ‘Receive Goods’), (‘Buy Goods’, ‘Verify Invoice’) and (‘Receive Goods’, ‘Verify Invoice’). In model 2 and 3 there are fewer matches or have weaker similarity with respect to their matches (e.g., ‘Buy Goods’, ‘Buy Special Goods Online’ and ‘Purchase Commodities’ are similar but not identical labels). In model 4 and 5 there are an even weaker match or no match at all with respect
to the features that they have in common with the query model. In the example, the technique in this thesis returns *model 1, 2 and 3* and discards *model 4 and 5* (detailed steps of process similarity search are given in Chapter 6).

![Diagram](image)

**Figure 4.3:** Searching a Collection of Business Process Models

Business process similarity search techniques have been developed from different perspectives [13, 35]. These techniques mainly vary with respect to the focuses and functions that they use to determine process similarity, as shown in Table 4.2, which compares ten selected process similarity search techniques besides our technique. All techniques providing indexes are selected. All techniques using feature-based similarity or edit distance as the function to measure process similarity are selected. For techniques using other functions, one technique for each function is selected for the purpose of comparison. Of the ten techniques, only two [44, 48] do not consider process definitions.

To the best of our knowledge, the technique described in [47] is the only one providing an index for process similarity search besides the technique described in this thesis, which combines M-Tree [12] with process similarity functions based on graph edit distance to reduce comparison operations. They compare process similarity based on complete process models, while we estimate process similarity based on different types of features. Furthermore, this technique completely relies on M-Tree, which requires that the process similarity function must be a metric, satisfying the properties of positivity, symmetry, and triangle inequality postulates.
### Table 4.2: Comparison of Process Similarity Search Techniques

<table>
<thead>
<tr>
<th></th>
<th>Focus*</th>
<th>Function*</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>This Thesis</td>
<td>Def</td>
<td>FS,ED</td>
<td>Structure-based</td>
</tr>
<tr>
<td>Dijkman et al. [30]</td>
<td>Def</td>
<td>ED</td>
<td>No</td>
</tr>
<tr>
<td>Li et al. [55]</td>
<td>Def</td>
<td>ED</td>
<td>No</td>
</tr>
<tr>
<td>Minor et al. [70]</td>
<td>Def</td>
<td>ED</td>
<td>No</td>
</tr>
<tr>
<td>Ehrig et al. [37]</td>
<td>Def</td>
<td>SS</td>
<td>No</td>
</tr>
<tr>
<td>Kunze et al. [47]</td>
<td>Def</td>
<td>ED</td>
<td>M-Tree</td>
</tr>
<tr>
<td>Grigori et al. [44]</td>
<td>Exe</td>
<td>ED</td>
<td>No</td>
</tr>
<tr>
<td>Kunze et al. [48]</td>
<td>Exe</td>
<td>JC</td>
<td>No</td>
</tr>
<tr>
<td>Dijkman et al. [31]</td>
<td>Def,Exe</td>
<td>ED,CF</td>
<td>No</td>
</tr>
<tr>
<td>Aalst et al. [3]</td>
<td>Def,Trace</td>
<td>PC</td>
<td>No</td>
</tr>
<tr>
<td>Lu et al. [59]</td>
<td>Def,Trace</td>
<td>FS</td>
<td>No</td>
</tr>
</tbody>
</table>

* Def=Definition, Exe=Execution.

ED=Graph Edit Distance, FS=Feature-based Similarity, SS=Semantic Similarity, SF=Similarity Flooding, JC=Jaccard Coefficient, CF=Causal Footprints, PC=Process Conformance.

Only three out of eleven functions summarized in [48] satisfy these properties. Although we also use M-Tree as one of the indexes for labels in this thesis, it is optional. For example, we can use the inverted index to index labels for the label similarity methods considering synonyms.

One technique [59], besides the technique described in this thesis, uses features of process models to measure process similarity. However, the goal of the technique [59] differs from the goal of this thesis (they want to measure similarity exactly), their features are larger than ours, potentially consisting of a complete process model. This makes their features suitable for measuring similarity exactly, but not for estimating it quickly.

Six techniques [30, 31, 47, 55, 70, 70], besides the technique described in this thesis, use graph edit distance to measure process similarity. Five of these six techniques (except for [47]) focus on defining a method to compute the similarity between two process models. However, to rank business process models in a collection, the similarity of each of the process models to the query model must be computed. Subsequently, the process models must be ordered according to their similarity. At the same time, business process model collections are increasing in
size. For example, Suncorp-Metway Ltd \[52\] maintains a collection of more than 6000 business process models. Comparing a query model with such amount of models is time consuming and can cause a similarity search operation to take multiple seconds or even minutes \[30\], depending on the method and algorithm that is used; while a query should be performed within milliseconds by a search engine. (Compare, for example, the response time that you would require of an Internet search engine.) Therefore, the technique described in this thesis complements existing business process similarity search techniques, because it focuses on estimating business process similarity, rather than measuring it exactly, and using that estimate to improve the time performance of existing techniques. As such it can be combined with any of the existing techniques to improve their performance.

Five other methods, besides features and graph edit distance, have been proposed for measuring process similarity, including semantic similarity, e.g., \[37\], similarity flooding, e.g., \[61\], Jaccard coefficient, e.g., \[48\], causal footprints, e.g., \[31\], and process conformance, e.g., \[3\].

\section*{4.4 Related Work}

This section presents the related work in other research areas, including general graph retrieval, schema matching, and ontology matching.

General graph retrieval has been applied in various application domains, including fingerprint, DNA and chemical compound retrieval. In these domains, (sub)graph isomorphism algorithms are used as a basis of graph retrieval, by checking whether a query graph is a subgraph of a graph in the dataset. To avoid comparing two entire graphs, which is time consuming, graph fragments are used as features to construct an index. This idea is also the basis for this thesis. Willett et al. \[93\] describe a feature-based approach to retrieve graphs in a chemical compound databases. ShaSha et al. \[75\] propose a path-based approach; Yan et al. \[95\] use discriminative frequent structures to index graphs; Zhao et al. \[106\] prove that using tree structures and a small number of discriminative graph structures to index graphs is a good choice. Yan et al. \[96\] also investigate the relation between feature-based and structure-based methods and built a connection between the two.

The main difference between the work that has been done in this area and the work in this thesis, is the different nature of business process graphs as compared to
4.4. Related Work

graphs in other domains. In particular, there is practically no restriction to the number of possible node labels in a business process graph and matching nodes do not necessarily have the identical labels. In comparison DNA nodes have four possible labels, chemical compound nodes have 117 possible labels, and in both cases matching nodes have identical labels. Also, business process graphs have different structural properties and patterns. These characteristics require that feature types are defined specifically for business process graphs. In addition to that processing feature similarity is different, because business process graphs do not require features to match exactly for graphs to be similar, while graphs in other domains do require features to match exactly.

The problem of process retrieval can be related to that of schema matching [74]. There are, however, important differences between process models and schemas. Firstly, data models and schemas generally have labeled edges (associations or schema elements) in addition to labeled nodes. Secondly, the types of nodes and the attributes attached to nodes are different in process models when compared to schemas or data models (e.g. there are no control nodes in data models). During our experiments, we implemented a graph matching technique originally designed for schema matching, namely Similarity Flooding [68]. After adapting the technique to deal with process models, we tested it on the dataset discussed in this thesis using various parameter settings [31]. The similarity flooding technique led to a poor score – 0.56 of mean average precision for the best settings (with a first-10 precision of 0.6). We attribute this poor performance to the fact that edges in process models do not have labels, while schema matching techniques, such as similarity flooding, heavily rely on edge labels. Madhusudan et al. [61] introduce a structural method for process model comparison based on similarity flooding. However, Madhusudan et al. rely on a semantic notation in which process models have labels attached to their edges.

The problem of process retrieval can also be related to that of ontology matching [39]. However, the nature of ontologies and business processes if different; a process model consists of labeled tasks and control flow relations, while an ontology provides a vocabulary, which records the relation of its terms, e.g., generalization and specialization. This makes it hard to directly use techniques from the area of ontology matching in the area of process retrieval. However, in future work, it would be worthwhile to investigate the possible use of ontologies and ontology matching
for matching tasks and task labels. Ehrig et al. [37] apply such a technique, using WordNet synonyms [69] as an ontology. In previous work we also applied WordNet synonyms to measure the semantic similarity of two labels [31, 34]. However, we still need to develop indexing techniques to use those similarity methods efficiently.
Chapter 5

PFIndex: a Process Feature Index

5.1 Introduction

Chapter 4 provides an introduction of process retrieval and identifies that there is no process-specific index to enhance its efficiency. Therefore, in this chapter, a process feature index (PFIndex) is proposed. A PFIndex is constructed using features, i.e., fragments of process models. As such, it retrieves process models by matching small fragments instead of the entire model, which reduces the computational complexity of process retrieval.

The remainder of the chapter is organized as follows. Section 5.2 abstracts process graphs from process models, which makes a PFIndex applicable to different (graph-based) process modeling languages. Section 5.3 defines process features in the context of process graphs. Section 5.4 constructs a PFIndex using process features.

5.2 Process Graph

The indexing technique is described in the context of process graphs. As defined in Definition 5.1, a process graph is a graph-based representation of a process model. The benefit of using a graph-based representation is that it can be used to represent
Chapter 5. PFIndex: a Process Feature Index

the structure of (graph-based) business process modeling languages. In this way, techniques that are defined for process graphs can be generically applied to models that are constructed with multiple business process modeling languages. As an example, Figure 5.1 shows the business process graphs of the models in Figure 4.2. The transformation is made by mapping events, activities, and gateways to nodes, and mapping sequence flows to edges. This thesis focuses on structures of process models, therefore BPMN elements, e.g., data objects and message flows, are not considered. As shown in graph 4 of Figure 5.1, we assign each type of routing elements (e.g., BPMN gateways and EPC connectors) a unique label to represent its routing function, e.g., ‘Xor-Split’ and ‘And-Join’. By doing this, routing elements in process graphs are labeled uniformly regardless of the notations of its original model.

Definition 5.1 (Process Graph, Pre-set, Post-set). Let \( \mathcal{L} \) be a set of labels. A process graph describes a (business) process as a tuple \((N, E, \lambda)\), in which:

- \( N \) is the set of nodes.
- \( E \subseteq N \times N \) is the set of edges.
- \( \lambda: N \rightarrow \mathcal{L} \) is an injective function that maps nodes to labels.

Let \( G = (N, E, \lambda) \) be a process graph and \( n \in N \) be a node: \( \bullet n = \{ m | (m, n) \in E \} \) is the pre-set of \( n \), while \( n\cdot = \{ m | (n, m) \in E \} \) is the post-set of \( n \).

There are limitations of representing a process model by a process graph defined in Definition 5.1. Firstly, it focuses on the activity and control flow aspects, however, eleven aspects are identified in Chapter 2. As such, it only supports part of the notational elements. Taking the core set BPMN elements (Figure 5.2) as an example, it transforms flow objects into nodes and transforms connecting objects into edges, while swimlanes and artifacts are not supported. Secondly, it does not distinguish different types of nodes and edges. Events, activities, and gateways of BPMN models are all transformed into nodes of process graphs; Sequence and message flows are all transformed of BPMN models into edges of process graphs. As such, a process graph cannot be transformed back into the original model. A more advanced version of process graph is defined in [31], which considers other aspects as attributes and also distinguishes different types of nodes and edges. The version in [31] can be easily adapted here if necessary. We use the simpler version (Definition 5.1) here since it is already sufficient to explain the concept in this part.
5.3 Features

In this section, features of process models are introduced. Features are defined as simple but representative abstractions of business process models. Their simplicity allows that the retrieval based on features is fast and their representativeness ensures that retrieval based on them is effective. This makes features very suitable as means to quickly retrieve business process models satisfying users’ requirements. Provided that we choose business process model features carefully, we can further speed up process retrieval by constructing an index of business process models based on those features. In this section, we present the business process model features that we explore in this thesis.

Labels can be conveniently used as features, because they are simple strings and therefore qualify as simple abstractions. In addition, indexing mechanisms for strings are well-known, which enables indexing of label features. However, it is harder to use the structure of a business process model as a feature. In fact, match-
Chapter 5. PFIndex: a Process Feature Index

ing the structure of business process models in previous work is what makes the problem computationally hard. Therefore, we consider the structure of a business process model in terms of the simpler structural features: sequence, split, join, and loop. Furthermore, we only consider small structures that contain edges no more than a threshold that can be set as desired, as defined in Definition 5.2. We define these features on the abstraction of a business process graph. We name these features basic features.

**Definition 5.2 (Basic Feature).** Let \( D \) be a collection of process graphs and \( g \in D \) be a process graph. A feature \( f \) of \( g \) is a subgraph of \( g \). The size of a feature is the number of edges it contains, denoted as \( \text{Size}(f) = |E_f| \). Let \( \text{max} \) be a threshold, indicating the maximal size of a feature that is considered. The type of a feature is the structural pattern of a feature, including, denoted \( \text{Type}(f) \in \{\text{node, sequence, split, join, loop}\} \). Feature \( f \) is

- a node feature consisting of node \( n \), if and only if \( E_f = \emptyset \) (its size is 0).
- a sequence feature of size \( s - 1 \) consisting of nodes \( \{n_1, n_2, \ldots, n_s\} \), if \( E_f \) is the minimal set containing \( (n_1, n_2), (n_2, n_3), \ldots, (n_{s-1}, n_s) \), for \( s \geq 2 \). It is denoted as \( n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_s \).
- a split feature of size \( s \) consisting of a split node \( n \) and a set of nodes \( \{n_1, n_2, \ldots, n_s\} \), if and only if \( E_f \) is the minimal set containing \( (n, n_1), (n, n_2), \ldots, (n, n_s) \), for \( s \geq 2 \). It is denoted as \( n \rightarrow \{n_1, n_2, \ldots, n_s\} \).
- a join feature of size \( s \) consisting of a join node \( n \) and a set of nodes \( \{n_1, n_2, \ldots, n_s\} \), if and only if \( E_f \) is the minimal set containing \( (n_1, n), (n_2, n), \ldots, (n_s, n) \), for \( s \geq 2 \). It is denoted as \( \{n_1, n_2, \ldots, n_s\} \rightarrow n \).
- a loop feature of size \( s \) consisting of nodes \( \{n_1, n_2, n_3, \ldots, n_s\} \), if \( E_f \) is the minimal set containing \( (n_1, n_2), (n_2, n_3), \ldots, (n_{s-1}, n_s), (n_s, n_1) \), for \( s \geq 1 \). It is denoted as \( n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_{s-1} \rightarrow n_s \).

The sequence, split, join, and loop features are referred as structural features.

The function \( \Gamma \) returns the set of features in a process graph \( g \), i.e., \( \Gamma(g) = \{f \text{ is a subgraph of } g | \text{Type}(f) \in \{\text{node, sequence, split, join, loop}\}\} \). Let \( \text{max} \) be a threshold of the size of a feature, \( \Gamma(g, \text{max}) = \{f \in \Gamma(g) | \text{Size}(f) \leq \text{max}\} \).
Figure 5.3 shows the features of graph 1-5 in Figure 5.1. The node feature set is \{'Order Goods', 'Order Goods Online', 'Receive Goods', 'Xor-Split', 'Pay', 'Receive Application', 'Approve'\} (labels are used to represent nodes here); the sequence feature set is \{'Order Goods'→'Receive Goods', 'Order Goods Online'→'Receive Goods', \ldots, 'Receive Goods'→'Order Goods'→'Xor-Split'\}; the split feature set is 'Xor-Split'→\{'Receive Goods', 'Pay'\}; and the loop feature set is \{'Order Goods'→'Xor-Split'→'Receive Goods'\}.

5.4 Process Feature Index (PFIndex)

As shown in Figure 5.3 there are already 29 features (22 different ones) for 5 small process graphs. Comparing each query feature to all these features is not efficient. To solve this issue, an index, called process feature index (PFIndex), is constructed using features of a collection of process models.

A PFIndex is a ‘parent-child’ index that exploits the fact that features of a larger size (in terms of the number of edges) are composed of features of a smaller size. For example: sequence features of size 1 are composed node features (of size 0); sequence features of size 2 are composed of sequence features of size 1 and, indirectly, of node features (of size 0). We call the (larger) composed features ‘child’ features and the (smaller) component features ‘parent’ features, as defined in Definition 5.3.

**Definition 5.3 (Parent Feature, Child Feature).** If a feature $f$ can generate feature $cf$ by adding a single edge and at most one node, feature $f$ is a direct parent feature of feature $cf$ and feature $cf$ is a direct child feature of feature $f$. It is denoted as $f \in DPFS(cf)$ or $cf \in DCFS(f)$, where $DPFS$ ($DCFS$) is a function that maps a feature to its direct parent (child) feature set. The parent and the child relation are the transitive closure of the direct parent and the direct child relation.

A PFIndex consists of a directed graph, in which each node corresponds to a feature and edges relate each feature to its direct children. Node features are the smallest possible features and, as a consequence, are not the child of any feature. The PFIndex maps each feature to the process graphs of which it is a feature and each node of a feature to the process graph nodes that it represents.

More precisely, a PFIndex is defined as follows.
Definition 5.4 (Process Feature Index (PFIndex)). Let $D$ be a collection of process graphs with disjoint sets of nodes. The process feature index of $D$, denoted $PFIndex(D)$, is a tuple $(F, RF, \nu)$, in which:
5.4. Process Feature Index (PFIndex)

- $F$ is the set of basic features of process graphs in $\mathcal{D}$.
- $RF = \{((f_1, f_2)) | f_1, f_2 \in F \land f_1 \in DPFS(f_2)\}$ is the direct parent-child relation between features as defined in Definition 5.3.
- $\upsilon : (\bigcup_{f \in F} N_f) \rightarrow \mathcal{P}(\bigcup_{g \in \mathcal{D}} N_g)$ is the function that maps each feature node to the graph nodes from which the feature is derived.

Algorithm 1 presents the algorithm to construct and manage a PFIndex. Constructing a PFIndex consists of the following two steps as shown in Algorithm 1:

1. Firstly, an empty index is initialized.
2. Then, each process graph in the collection is inserted into the index.

**Algorithm 1:** Construct a PFIndex

**input:** a collection of process graphs: $\mathcal{D}$

**output:** a PFIndex: $PFIndex = (F, RF, \upsilon)$

1 begin
2 $PFIndex \leftarrow$ null;
3 foreach $g \in \mathcal{D}$ do
4 insertGraph ($g$, $PFIndex$); // Algorithm 2
5 return $PFIndex$;

Inserting a process graph into a PFIndex is described in Algorithm 2. Features of the process graph are generated before being inserted into the index. These features are inserted in order, i.e., parent features before child features. This is because a child feature must connect to its parent features, which requires the parent features are already in the PFIndex. To insert a feature, there are two possibilities. If the feature is in the index, only the nodes, that the feature is derived from, is inserted ($\upsilon$). If the feature is not in the index, the feature is created ($F$) and inserted into the index by relating with its parent features in the index ($RF$); the nodes that the feature is derived is inserted ($\upsilon$).

Deleting a process graph from a PFIndex is described in Algorithm 3. Features of the graph are also deleted in order, i.e., child features before parent features. This is because that a child feature requires its parent features to be located through their relations in the index (detailed steps about locating a feature in a PFIndex are given in Chapter 6 when presenting feature matching through a PFIndex). Similarity to
Algorithm 2: Insert a process graph into a PFIndex

function insertFeature ($\mathcal{F}$, PFIndex) begin

while $\mathcal{F} \neq \emptyset$ do

foreach $f \in \mathcal{F}$ do

if $\exists f_1 \in \mathcal{F}, f_1 \in DPFS(f) //Definition 5.3$ then

if $f \notin \mathcal{F}$ then

$F \leftarrow F \cup \{f\}$;

foreach $f_1 \in DPFS(f) \subseteq \mathcal{F} //Definition 5.3$ do

$RF \leftarrow RF \cup \{(f_1, f)\}$;

$v(N_f) \leftarrow v(N_f) \cup N_f$;

$F \leftarrow F - \{f\}$;

end

end

end

end

input: a process graph: $g$, a PFIndex: PFIndex = ($\mathcal{F}$, $RF$, $v$), a threshold: $max$

output: a PFIndex: PFIndex = ($\mathcal{F}$, $RF$, $v$)

begin

$\mathcal{F} = \Gamma(g, max) //Definition 5.2$

insertFeature ($\mathcal{F}$, PFIndex);

return PFIndex;

end

inserting a graph, for each feature to be deleted, there are also two possibilities. If the feature in the index also maps to other features besides the feature to be deleted, the nodes that the feature is derived is deleted ($v$). Otherwise, the corresponding feature ($F$) and its relations with its parent features ($RF$) are deleted in the index.

As described in Algorithm 4, updating a process graph into a PFIndex is done by deleting features in the original process graph but not in the updated process graph and inserting features in the updated process graph but not in the original process graph.

Let $k$ be the total number of process models in a collection; $n$ and $m$ be the average number of nodes and edges in a process model; $max$ be the maximal size of a feature that is considered to construct a PFIndex. The number of features
Algorithm 3: Delete a process graph from a PFIndex

1. function deleteFeature (F, PFIndex)
2. begin
3. while F ≠ ∅ do
4.   foreach f ∈ F do
5.     if ∄ f₁ ∈ F, f₁ ∈ DCFS(f) //Definition 5.3 then
6.       if v(f) = {N_f} then
7.         F ← F − {f};
8.       foreach f₁ ∈ DCFS(f) ∩ F //Definition 5.3 do
9.         RF ← RF − {(f₁, f)};
10.     else v(N_f) ← v(N_f) − {N_f} ;
11.     F ← F − {f};
end

input: a process graph: g, a PFIndex: PFIndex = (F, RF, v),
a threshold: max
output: a PFIndex: PFIndex = (F, RF, v)

Algorithm 4: Update a process graph in a PFIndex

input: the original process graph: g, the updated process graph: g₁,
a PFIndex: PFIndex = (F, RF, v), a threshold: max
output: a PFIndex: PFIndex = (F, RF, v)

begin
1. F = Γ(g, max);//Definition 5.2
2. F₁ = Γ(g₁, max);//Definition 5.2
3. deleteFeature (F − F₁, PFIndex); //a function in Algorithm 3
4. insertFeature (F₁ − F, PFIndex); //a function in Algorithm 2
5. return PFIndex;
in a process graph (that are considered) are no more than \( n + \sum_{1 \leq i \leq max} C_m^{max} \). Therefore, both time and space complexities for managing (inserting, deleting, and updating) a PFIndex are \( O(n + \sum_{1 \leq i \leq max} C_m^{max}) = O(n + m^{max}) \); both time and space complexities for constructing a PFIndex are \( O(k \cdot (n + m^{max})) \). The threshold \( max \) is a small number, e.g., from 0 to 3 in our experiments in Chapter 8.

Figure 5.4 presents the PFIndex of graph 1-3 (and their features) in Figure 5.3. Node feature ‘Receive Goods’, derived from graph 1-3, is a direct parent feature of sequence feature ‘Order Goods’→‘Receive Goods’, derived from graph 1, 3; sequence feature ‘Order Goods’→‘Receive Goods’ is a direct child feature of node feature ‘Order Goods’. Figure 5.5 presents the updated PFIndex by inserting features of graph 4, 5. By comparing these two indexes, we can see that, for features that appear in Figure 5.4, the nodes that these features are derived from, are updated. For example, in Figure 5.5 node feature ‘Receive Goods’ is also derived from graph 4. For features that do not appear in Figure 5.4, new nodes representing these features are created and connected with existing parent and child features. For example, sequence feature ‘Receive Goods’→‘Order Goods’ is connected node feature ‘Receive Goods’. New features are not connected to existing features if there are no parent-child relations between them. For example, node feature ‘Approve’ and sequence feature ‘Receive Application’→‘Approve’ in Figure 5.5 are not connected with features that appear in Figure 5.4.

For simplicity, ids are used to represent nodes: 1 = order goods (online), 2 = receive goods, 3 = pay.

**Figure 5.4**: Process Feature Index of graph 1-3
A PFIndex helps to increase the efficiency of retrieval, if we consistently start by matching ‘smaller’ features, keeping track of those matches and subsequently, when matching ‘larger’ features, re-using the matches of the ‘smaller’ features that were already identified. For example, suppose that we need to search for models that are matching with a model that consists of only a single sequence: ‘Order’→‘Receive’. Suppose that node feature ‘Order’ matches with node feature ‘Order Goods’, and node feature ‘Receive’ matches with node feature ‘Receive
Chapter 5. PFIndex: a Process Feature Index

Goods’ (metrics for feature matching are defined in Section [6]). Then to match sequence feature ‘Order’→‘Receive’ through PFIndex, only the common child features of node feature ‘Order Goods’ and ‘Receive Goods’ are required to be checked. In this case, sequence feature ‘Order Goods’→‘Receive Goods’ need to be checked in stead of all sequence features and it matches with the query sequence. More details about retrieving through a PFIndex is provided in Chapter [6].

A drawback of a PFIndex is that it does not help retrieve node features more efficiently. Therefore, it requires information retrieval indexing techniques to retrieve node features more efficiently, e.g., the inverted index and M-Tree. More details about integrating a PFIndex with these indexes is provided in Chapter [6].
Chapter 6

Applying PFIndex to Process Retrieval

6.1 Introduction

Chapter 5 presents a process feature index, PFIndex, by connecting features based on their ‘parent-child’ relations. In this chapter, the PFIndex is applied to process retrieval (both process querying and similarity search) to enhance its efficiency. Instead of directly matching a given query (search) process model with process models in a collection, in this chapter, features of the query (search) process model are identified and queried through a PFIndex, which consists of features of process models in a collection and parent-child relations between features. Then process models that match the query (search) process graph are retrieved and checked based on the matched features.

The remainder of the chapter is organized as follows. Section 6.2 presents how to match nodes between two process graphs, which is a basis for feature matching. Section 6.3 presents steps to perform process querying through a PFIndex. Section 6.4 presents steps to perform process similarity search through a PFIndex; it also presents an improved greedy algorithm for process similarity search to further enhance the efficiency.
6.2 Node Feature Matching

To perform process retrieval [30], the first step is to match node features from a query (search) model to node features of models in a collection. After matching node features, structural features of process models can be further compared, which is explained latter in Section 6.3 and 6.4. This section presents the aspects that can be used to matching node features and provides functions that consider the activity or control flow aspects for illustrative purposes.

Node matching can be measured using different process aspects (eleven aspects are defined in Chapter 2, e.g., data and resource). For example, Dijkman et al. propose five functions to measure node similarity, based on syntax, semantics, attribute, type, and context [31]. The syntactic function considers the syntax of node labels, which focuses on the activity aspect; the semantic functions considers the semantics of the words in the labels, which focuses on both the activity and semantic aspects; the type function considers the node types, which also focuses on the activity aspect; the contextual function considers the context of the nodes, which focuses on the control flow aspect; the attribute function consider attribute values, which focuses on multiple aspects, since any aspect, except for activity and control flow, can be an attribute, e.g., resource, organization, etc.

In this thesis, two most common aspects, the activity and control flow aspects, are considered (Table 3.1). However, other aspects can be considered by replacing these node matching functions [31]. Three functions are presented in this section, two of them based on node labels and one of them based on node contexts.

The activity aspect is considered by comparing node labels in this thesis, which are denoted as label features. In the text retrieval field, a common function is to retrieve text files that contain all the words in a query (search) text file [64]. The same concept can be applied to label retrieval as defined in Definition 6.1. Lower case versions of the words are used for comparison and stop-words, e.g., ‘a’, ‘an’, ‘the’, ‘one’, . . . are not considered.

**Definition 6.1** (Word-based Label Feature Matching). Let \( l_q \) and \( l \) be two label features. Let \( \omega(l) \) be the function returns the set of words that appear in a label \( l \). Given \( l_q \) a query (search) label feature, label feature \( l \) is retrieved if and only if label feature \( l \) contains all the words in label \( l_q \), i.e., \( \omega(l_q) \subseteq \omega(l) \).

For example, ‘Receive Goods’ and ‘Receive Application’ are matching features
for a query (search) ‘Receive’.

An inverted index \[64\] can be constructed to more efficiently retrieve matching label features in a process model collection for a given query label feature, using Definition 6.1. For each word in a query label (feature), the set of label features containing the word can be quickly retrieved through an inverted index, and the intersection of these sets contains all matching label features for the query label feature. We refer to \[64\] for more details about indexing and retrieving labels (texts) with an inverted index.

The drawback of the word-based label feature matching is that it returns a true/false result, i.e., whether two label features are matching or not. This is sufficient for process querying, however, for process similarity search, the similarity of two label features may be required \[30\]. Definition 6.2 presents a function that results in a similarity score between 0 and 1 for two label features \[30\].

**Definition 6.2 (Distance-based Label Feature Similarity).** Let $G = (N, E, \lambda)$ be a business process graph and $n, m \in N$ be two node features and let $|l|$ represent the number of characters in a label $l$. The string edit distance of the label features $\lambda(n)$ and $\lambda(m)$ of the nodes, denoted $\text{ed}(\lambda(n), \lambda(m))$ is the minimal number of atomic string operations needed to transform $\lambda(n)$ into $\lambda(m)$ or vice versa. The atomic string operations are: inserting a character, deleting a character or substituting a character for another. The label feature similarity of $\lambda(n)$ and $\lambda(m)$, denoted $\text{lsim}(n, m)$ is:

$$\text{lsim}(n, m) = 1.0 - \frac{\text{ed}(\lambda(n), \lambda(m))}{\max(|\lambda(n)|, |\lambda(m)|)}$$

The distance-based label feature similarity can also be used to measure label feature matching by setting a cutoff parameter. Two labels are considered to be matching if their similarity score is bigger than the cutoff.

For example, the string edit distance between ‘Transportation planning and processing’ and ‘Transporting’ is 24: delete ‘ion planning and process’. Consequently, the label feature similarity is $1.0 - \frac{24}{38} \approx 0.37$. Supposing 0.6 is set as the cutoff, these two label features are not matching. Optional preprocessing steps, such as lower-casing and removing special characters, can improve the results of feature similarity measurements.

An M-Tree index \[12\] can be constructed to more efficiently retrieve matching label features in a process model collection for a given query label feature, using
Chapter 6. Applying PFI\textsuperscript{ndex} to Process Retrieval

Definition 6.2. An M-Tree can help quickly retrieve all label features that have a bigger similarity score with a query label feature than a given cutoff. We refer to [12] for more details about indexing and retrieving labels with an M-Tree.

The drawback of measuring node matching only by labels is that similar activities (node features) can have different labels. Therefore, it may be hard to matching node features solely based on label features. For example, in Fig. 6.8, ‘Buy Special Goods Online’ of Graph 2 and ‘Purchase Commodities’ of Graph 3 are related to ‘Buy Goods’ of Query graph. However, compared with ‘Buy Goods’, ‘Buy Special Goods Online’ is more verbose and ‘Purchase Commodities’ uses synonyms. Therefore, they may not be matching based on the label feature. To deal with this situation, we consider contextual information of nodes, i.e., the control flow aspect.

We can measure the contextual similarity of two nodes, by determining the similarity of the (structural) roles that they have in their business process graphs. We distinguish five different roles that nodes can have: start, stop, regular(sequence), split or join, as defined in Definition 6.3. We do not distinguish the type of splits or joins (e.g., XOR or AND), which are distinguished in their label features.

**Definition 6.3 (Role Feature).** Let $n \in N$ be a node and $\mathcal{R} = \{\text{start, stop, split, join, regular}\}$ be a set of roles that a node can have. The roles of $n$ are determined by the function $\text{roles} : N \rightarrow \Pi(\mathcal{R})$, such that

\[
\begin{align*}
\text{start} \in \text{roles}(n) & \iff |\bullet n| = 0 \\
\text{stop} \in \text{roles}(n) & \iff |n \bullet| = 0 \\
\text{split} \in \text{roles}(n) & \iff |n \bullet| \geq 2 \\
\text{join} \in \text{roles}(n) & \iff |\bullet n| \geq 2 \\
\text{regular} \in \text{roles}(n) & \iff |\bullet n| = 1 \land |n \bullet| = 1
\end{align*}
\]

Roles of nodes are considered to be similar or not with respect to the input and output paths of the nodes. The definition of role feature similarity is inspired by string edit-distance, i.e., mainly considering the differences between numbers of input (output) paths of two nodes. Formally, role feature similarity is defined as follows:

**Definition 6.4 (Role Feature Similarity).** Let $n, m \in N$ be two nodes. The role
6.2. Node Feature Matching

The feature similarity of these two nodes, denoted $\text{rsim}(n, m)$, is defined as:

$$
\text{rsim}(n, m) = \begin{cases} 
1 & \text{if } \text{start} \in \text{croles} \land \text{stop} \in \text{croles} \\
\frac{\text{avg}(1 - \frac{|\text{\textbullet n}| - |\text{\textbullet m}|}{|\text{\textbullet n}| + |\text{\textbullet m}|}, 1)}{\text{abs}(|\text{\textbullet n}| - |\text{\textbullet m}|)} & \text{if } \text{start} \in \text{croles} \land \text{stop} \notin \text{croles} \\
\frac{\text{avg}(1, 1 - \frac{|\text{\textbullet n}| - |\text{\textbullet m}|}{|\text{\textbullet n}| + |\text{\textbullet m}|})}{\text{abs}(|\text{\textbullet n}| - |\text{\textbullet m}|)} & \text{if } \text{start} \notin \text{croles} \land \text{stop} \in \text{croles} \\
\frac{1 - \text{avg}(1 - \frac{|\text{\textbullet n}| - |\text{\textbullet m}|}{|\text{\textbullet n}| + |\text{\textbullet m}|})}{\text{abs}(|\text{\textbullet n}| - |\text{\textbullet m}|)} & \text{otherwise}
\end{cases}
$$

Where $\text{croles} = \text{roles}(n) \cap \text{roles}(m)$.

This formula covers all possible combinations of roles that nodes can have. For example, the situation in which both nodes are split nodes as well as join nodes is covered by the case ‘otherwise’ (start $\notin \text{croles} \land \text{stop} \notin \text{croles}$). The situation in which both nodes are regular nodes is covered by the same case and leads to a role feature similarity score of 1.

For example, the role feature similarity of a node with no input edges and two output edges and a node with no input edges and one output edges is $(1 + (1 - 1/3))/2 = 0.67$.

The drawback of measuring role similarity in this way is that it does not discount for the fact that there is a large difference between the frequency of the occurrence of the different role features. Therefore, using the role similarity metric in this way is ineffective. Since, if we give a bonus for matching role features, most nodes would receive that bonus. Therewith, the effect of the bonus would be minimal.

For that reason we refine the role similarity metric to take this effect into account. We do that by not considering features that appear too frequently in the dataset; we say that those features lack ‘discriminative power’.

**Definition 6.5 (Discriminative Role Features).** Let $\mathcal{N}$ be the set of nodes of all business process models in a collection. We say that a role feature $r \in \mathcal{R}$ is discriminative, denoted $\text{discriminative}(r)$ if and only if the fraction of the nodes that have the feature is sufficiently small:

$$
\left| \left\{ n | n \in \mathcal{N}, r \in \text{roles}(n) \right\} \right| \leq \text{dcutoff} \frac{|\mathcal{N}|}{|\mathcal{N}|}
$$

Where $\text{dcutoff}$ is a cutoff value that determines when the fraction of nodes that have the feature is sufficiently small. This cutoff value is a parameter that can be set as desired, to produce the best results.

1 Function $\text{avg}$ returns the average value; function $\text{abs}$ returns the absolute value.
In general, a good setting for $dcutoff$ is easy to determine, because there is a large difference between the frequency of features with a low frequency of occurrence and features with a high frequency of occurrence. For example, in the set of 100 SAP reference models that we use for evaluation in this thesis, there are 374 nodes in total (only counting EPC functions). Of these nodes, 178 have the ‘stop’ role, 153 have the ‘start’ role, 58 the ‘regular’ role, 52 the ‘split’ role, 36 the ‘join’ role. Here, we have far more nodes with the ‘start’ and ‘stop’ roles than other nodes. Hence, if we set the $dcutoff$ anywhere between 0.16 and 0.40, ‘start’ and ‘stop’ role features are not considered discriminative, while other role features are considered discriminative. We incorporate the discriminative power of role features into their similarity using the following formula.

**Definition 6.6 (Role Feature Similarity with Discriminative Power).** Let $n, m \in N$ be two nodes. Their role feature similarity with discriminative power, denoted $rdsim(n, m)$, is defined as:

$$rdsim(n, m) = \begin{cases} 
  \text{rsim}(n, m) & \text{if } \forall r \in \text{roles}(n) \cap \text{roles}(m) : \text{discriminative}(r) \\
  0 & \text{otherwise}
\end{cases}$$

The role feature similarity can also be used to measure role feature matching by setting a cutoff parameter. The role features of two nodes are considered to be matching if their similarity score is bigger than the cutoff.

For example, assuming that ‘start’ and ‘stop’ role features are not considered discriminative, then the role feature similarity of a node with no input edges and two output edges and a node with no input edges and one output edges is $(0 \times 1 + 1 \times (1 - 1/3))/2 = 0.17$.

An inverted index \cite{64} can be constructed to more efficiently retrieve matching role features in a process model collection for a given query role feature, using Definition 6.6. Using an inverted index, the same role features are not required to be compared with a query role feature multiple times. For example, there are 58 the ‘regular’ role features in the set of 100 SAP reference models that we use for evaluation in this thesis, and we only need to compute its role similarity with a given query role feature once.

We can measure the similarity of two node features using combinations of similarity functions. Definition 6.7 presents a function to compute the similarity of two node features by combining the distance-based label feature similarity (Definition 6.2) and role feature similarity (Definition 6.6). Strong label feature similarity...
is a strong indication that two nodes are matched, while a combination of role feature similarity and (less strong) label feature similarity is also an indication that two nodes are matched. We distinguish between these two cases when determining a node feature match, such that we can set different thresholds for label similarity in case there is also role similarity and in case there is no role similarity.

**Definition 6.7** (Node Feature Matching). Let $n, m \in N$ be two node features with their respective label features and role features. The node features are matching, if they satisfy one of the following two rules:

- their label features are similar to a high degree, i.e., $\text{lsim}(n, m) \geq \text{lcut}\text{off}_{\text{high}}$;

- their role features are similar, and their label features are similar to a medium degree, i.e., $\text{rdsim}(n, m) \geq \text{rcut}\text{off}$ and $\text{lsim}(n, m) \geq \text{lcut}\text{off}_{\text{med}}$.

Where $\text{lcut}\text{off}_{\text{high}}$, $\text{rcut}\text{off}$ and $\text{lcut}\text{off}_{\text{med}}$ are parameters that determine what is considered to be a similar to what degree. The parameters can be set as desired, to produce the best results.

To more efficiently retrieve matching node features in a process model collection for a given query node feature, using Definition 6.6, an M-Tree index [12] and an inverted index [64] can work together. The node retrieval is performed in the following steps. Firstly, an M-Tree helps quickly retrieve the set of node features having a label similarity score no less than $\text{lcut}\text{off}_{\text{high}}$ with respect the label of the query node feature. Secondly, an M-Tree helps quickly retrieve the set of node features having a label similarity score no less than $\text{lcut}\text{off}_{\text{med}}$ and less than $\text{lcut}\text{off}_{\text{high}}$ with respect the label of the query node feature. Thirdly, an inverted index helps quickly retrieve the set of node features having a role similarity score no less than $\text{lcut}\text{off}_{\text{med}}$ with respect to the roles of the query node feature. Fourthly, the intersection of the sets in the second and third steps unions the set in the first step, and the outcome set contains all matching node features of the query node feature.

More advanced functions can be used that take synonyms and stemming [31, 34] and domain ontologies [37] into account. However, involving these functions would increase the execution time (though improve the result quality), which is against the purpose of this thesis. Therefore, we keep the similarity functions simple to provide a more efficient querying technique with reasonable result quality (evaluations provided in Chapter 8).
6.3 Process Querying

Current process querying techniques mainly focus on the expressive power of the process query language, which is not efficient enough. For example, it on average takes 5s to run a query with a collection of 500 process models, using BPMN-Q [10]. While a search engine, e.g., Google, should respond within milliseconds. Therefore, this section presents how to perform process similarity search more efficiently. It compares features of process models instead of process models directly and retrieve potential matching models based on feature matching. It provides an index for features, PFIndex, using which only a small number of process features of the collection needs to be compared with query features. It checks contradictions for each retrieved model by checking the matching features instead of comparing the entire model with the query model using current techniques.

The process querying technique in this thesis works in three steps, as shown in Fig. 6.1. Firstly, a query model is transformed into a query process graph and query features are identified (Section 6.3.1). Secondly, query features are used to retrieve matching features through a PFIndex (Section 6.3.2). Thirdly, process graphs are retrieved and checked based on the matching features (Section 6.3.3).

![Figure 6.1: Steps of Process Querying](image)

Figure 6.2 presents five process graphs and Figure 6.3 presents four query process graphs. In the remainder of this section, these graphs are used to illustrate the process querying technique in this thesis.

6.3.1 Query Feature

To perform process querying, a query process model should be provided. A query process model can contain notational elements from the business process modeling notation in use. We refer to notational elements as basic elements. To make querying more powerful, advanced business process query languages [6, 14, 25] also use the following types of nodes and edges: wildcard node, transitive edge, neg-
6.3. Process Querying

Figure 6.2: A Collection of Process Graphs

Figure 6.3: A Collection of Query Process Graphs

ative edge, and neg-transitive edge. We refer to these four elements as advanced elements.

In this section, (to be consistent with process graphs) a query process model is transformed into a query process graph, as defined in Definition 6.8. A process graph consists of a tuple of a node set, an edge set, and a function maps a node to a label. Compared with a process graph, a query process graph has two more functions, one mapping a node to a node type and the other one mapping an edge to an edge type. The node types consist of basic and wildcard. A basic node is represented as a node with a label and a wildcard node is represented as a node without a label. For example, query c in Figure 6.3 contain two basic nodes (‘Receive’ and ‘Pay’) and a wildcard node (the one without a label). The edge types consist of basic, transitive, negative, and neg-transitive. A basic edge is represented as an edge without a label. A transitive edge is represented as an edge with ‘∗’ as its label. A negative edge is an edge with ‘¬’ as its label. A neg-transitive edge is represented as an edge with both ‘∗’ and ‘¬’ as its label. In Figure 6.3 query c contains a basic edge and a negative edge; query d contains a transitive edge and a neg-transitive edge.
Chapter 6. Applying PFIndex to Process Retrieval

**Definition 6.8 (Query Process Graph).** Let \( \mathcal{L} \) be a set of labels. A query process graph is a process graph that can contain advanced elements besides basic ones, defined as a tuple \( Q = (N, E, \lambda, \Theta, \theta) \), in which:

- \( N \) is the set of nodes.
- \( E \subseteq N \times N \) is the set of edges.
- \( \lambda : N \rightarrow \mathcal{L} \) is an injective function that maps a node to a label.
- \( \Theta : N \rightarrow \{\text{basic, wildcard}\} \) is an injective function that determines whether a node is a basic or a wildcard query node.
- \( \theta : E \rightarrow \{\text{basic, transitive, negative, neg-transitive}\} \) is an injective function that determines whether an edge is a basic, a transitive, a negative, or a neg-transitive edge.

Because a query process graph can also contain advanced elements, we consequently need advanced features to be able to break up a query process graph into query features as defined in Definition 6.9.

**Definition 6.9 (Query Feature).** Let \( Q = (N, E, \lambda, \Theta, \theta) \) be a query process graph. An query feature \( F = (N_F, E_F, \lambda, \Theta, \theta) \) of \( Q \) is a subgraph of \( Q \) \( (F \subseteq Q) \). The query feature is a:

- wildcard feature consisting of node \( n \), if and only if \( N_F = \{n\} \) and \( \Theta(n) = \text{wildcard} \) (its size is 0). It is denoted as "(an empty label).

- transitive feature is a sequence feature of size 1, consisting of nodes \( \{n_1, n_2\} \), if and only if \( E_F = \{(n_1, n_2)\} \) and \( \theta((n_1, n_2)) = \text{transitive} \). It is denoted as \( n_1 \rightarrow n_2 \).

- negative feature is a sequence feature of size 1, consisting of nodes \( \{n_1, n_2\} \), if and only if \( E_F = \{(n_1, n_2)\} \) and \( \theta((n_1, n_2)) = \text{negative} \). It is denoted as \( n_1 \nrightarrow n_2 \).

- neg-transitive feature is a sequence feature of size 1, consisting of nodes \( \{n_1, n_2\} \), if and only if \( E_F = \{(n_1, n_2)\} \) and \( \theta((n_1, n_2)) = \text{neg-transitive} \). It is denoted as \( n_1 \nrightarrow * n_2 \).

- basic feature, if and only if \( \forall e \in E_F, \theta(e) = \text{basic} \), and it is a feature according to Definition 5.2.
The wildcard, transitive, negative, and neg-transitive features are referred as advanced features. The function $\Gamma$ returns the set of query features in a query process graph $qg$.

Fig. 6.4 shows the features of query process graphs in Fig. 6.2. The node feature without a label is a wildcard feature, denoted as ‘.’. ‘Receive’ $\not\rightarrow$ ‘Pay’ is a negative feature. ‘Order’ $\rightarrow$ ‘Receive’ is a transitive feature. ‘Receive’ $\not\rightarrow$ ‘Pay’ is a neg-transitive feature. The others are basic features. Note that wildcard nodes can appear in a basic query feature, such as ‘’ $\rightarrow$ ‘Receive’. Matching query features with features of process graphs is explained in Section 6.3.2.

6.3.2 Feature Matching through Indexes

In Section 6.3.1 query features are introduced. This section presents how to match query features with features of process graphs. It first defines feature matching for each type of query feature and then presents how to match features through the indexes.

Two basic structural features are matching, if and only if their component features (node features) and edges between node features are matching, as defined in Definition 6.10. Basic features of different structures or sizes are never matching with each other.
Chapter 6. Applying PFIndex to Process Retrieval

Definition 6.10 (Basic Structural Feature Matching). Two sequence features of size \( s - 1 \), \( n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_s \) and \( m_1 \rightarrow m_2 \rightarrow \ldots \rightarrow m_s \), are matching if and only if for each \( 1 \geq i \geq s \): the node features, \( n_i \) and \( m_i \), are matching.

Two split features of size \( s \), \( n \rightarrow S_n \) and \( m \rightarrow S_m \) (\( S_n = \{n_1, n_2, \ldots, n_s\} \) and \( S_m = \{m_1, m_2, \ldots, m_s\} \)), are matching if and only if the node features, \( n \) and \( m \), are matching and there exists a mapping \( \text{Map} : S_n \rightarrow S_m \) holds that for each \( (sn, sm) \in \text{Map} \): the node features, \( sn \) and \( sm \), are matching. A join feature matching is defined similarly.

Two loop features of size \( s \) , \( n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_s \) and \( m_1 \rightarrow m_2 \rightarrow \ldots \rightarrow m_s \) are matching if and only if there exists a constant number \( 1 \leq c \leq s \), for each \( 1 \geq i \geq s \): the node features, \( n_k \) and \( m_{c+s-k} \), are matching.

An advanced query feature is matching with a process graph, if and only if the process graph has node features that are matching with the node features in the query feature and its structure between these matching nodes is matching with the edge in the query feature, as defined in Definition 6.10.

Definition 6.11 (Advanced Structural Feature Matching). Let \( G = (N, E, \lambda_G) \) be a process graph.

- a negative query feature, \( m_1 \not\rightarrow m_2 \), is matching with the process graph if and only if there exist \( n_1 \) and \( n_2 \) in \( N \), such that \( m_1 \) and \( n_1 \) (\( m_2 \) and \( n_2 \)) are matching and there exists no edge \( (n_1, n_2) \) in \( E \).

- a transitive query feature, \( m_1 \star \rightarrow m_2 \), is matching with the process graph if and only if there exist \( n_1 \) and \( n_s \) in \( N \), such that \( m_1 \) and \( n_1 \) (\( m_2 \) and \( n_s \)) are matching and there exists a basic sequence feature \( n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_{s-1} \rightarrow n_s \) in \( G \).

- a neg-transitive query feature, \( m_1 \not\star \rightarrow m_2 \), is matching with the process graph if and only if there exist \( n_1 \) and \( n_s \) in \( N \), such that \( m_1 \) and \( n_1 \) (\( m_2 \) and \( n_s \)) are matching and there exists no basic sequence feature \( n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_{s-1} \rightarrow n_s \) in \( G \).

Definition 6.1, 6.10, and 6.11 defines feature matching for different features. Definition 6.12 summarizes these definitions by mapping nodes and edges of features. Furthermore, the definition can also be used for matching between a query process graph and a process graph.

\(^2c+s-k\) returns the remainder of the plus of \( c \) and \( k \) divided by \( s \)
6.3. Process Querying

Definition 6.12 (Feature Matching/Process Querying). A business process graph $G = (N_G, E_G, \lambda_G)$ matches a query graph $Q = (N_Q, E_Q, \lambda_Q, \Theta_Q, \theta_Q)$, if and only if there exists a mapping $M : N_Q \rightarrow N_G$, such that:

- for each $(n_Q, n_G) \in M$, $n_Q$ is matching with $n_G$, as described in Section 6.2;
- if $(n_Q, m_Q) \in E_Q$ and $\theta(n_Q, m_Q) = \text{basic}$ then $(M(n_Q), M(m_Q)) \in E_G$;
- if $(n_Q, m_Q) \in E_Q$ and $\theta(n_Q, m_Q) = \text{negative}$ then $(M(n_Q), M(m_Q)) \notin E_G$;
- if $(n_Q, m_Q) \in E_Q$ and $\theta(n_Q, m_Q) = \text{transitive}$ then there exists a sequence $M(n_Q) \rightarrow \ldots \rightarrow M(m_Q)$ in $G$;
- if $(n_Q, m_Q) \in E_Q$ and $\theta(n_Q, m_Q) = \text{neg-transitive}$ then there does not exist a sequence $M(n_Q) \rightarrow \ldots \rightarrow M(m_Q)$ in $G$.

Let $f_Q$ and $f_G$ be two features. If $f_Q$ is matching with $f_G$, it is denoted $\text{match}(f_Q, f_G)$.


Algorithm 5 presents the algorithm to querying with basic features through the indexes. For a basic node, matching node features are retrieved as defined in Section 6.2 (line 16); for a wildcard feature, all node features are retrieved for the wildcard feature (line 13-14). Matching features for a basic structural feature are retrieved through a PFIndex. It first retrieves all features that are potentially matching with a given query feature based on parent-child relations (line 19-21), and then check whether the structures of potentially matches are matching with the
Algorithm 5: Basic Feature Retrieval

\begin{algorithm}
\begin{algorithmic}[1]
\Function{checkEdge}{qf,pqf,f,M}
\State \ForEach{$(qn_1, qn_2) \in (E_{qf} - E_{pqf})$}
\State \ForEach{$n_1 \in (M(qn_1) \cap N_f)$}
\State \ForEach{$n_2 \in (M(qn_2) \cap N_f)$}
\If{$(n_1, n_2) \in E_f$} \Return \text{True}; \EndIf
\EndFor
\EndFor
\EndFor
\State \Return \text{False};
\EndFunction

\textbf{input} : a query feature: $qf$, a PFIndex: $\text{PFIndex} = (F,RF,\nu)$,
a threshold: $max$, a mapping: $M : \Gamma(qg,\text{max}) \rightarrow \mathbb{P}(\bigcup_{g \in \mathcal{D}} \Gamma(g,\text{max}))$
\begin{footnotesize}
//\footnotesize$\Gamma(g,\text{max})$ is defined in Definition 5.2
\end{footnotesize}

\textbf{output}: a set of features: $RS$, the mapping: $M$
\begin{algorithmic}[1]
\State \If{$\exists (qf,F) \in M$} \Return $M(qf)$; //\footnotesize$F \in \bigcup_{g \in \mathcal{D}} \Gamma(g,\text{max})$ is a set of features \EndIf
\State \Else
\State \If{$|E_{qf}| == 0$} \If{$\forall n \in N_{qf}, \Theta(n) = \text{wildcard}$} \State $RS = \{f \in \bigcup_{g \in \mathcal{D}} \Gamma(g,\text{max})||N_f| = 1\}$; //all node features in $F$ \EndIf
\State \Else \State $RS$ is the set of matching node features of $qf$ (Section 6.2); \EndIf
\EndIf
\State \ElseIf{$|E_{qf}| > 0$ \&\& $\forall e \in E_{qf}, \theta(e) = \text{basic}$} \State $RS \leftarrow F$; \EndIf
\State \ForEach{$pqf \in \text{DPFS}(qf)$}
\State $PRS \leftarrow \text{featureRetrieval}(pqf, \text{PFIndex})$; //recursion
\State $RS \leftarrow RS \cap (\bigcup_{pf \in PFS} \{f \in RF(pf) | \text{checkEdge}(qf,pqf,f,M) == \text{True}\})$;
\EndFor
\State $M(qf) = RS$;
\State \Return $RS$;
\EndIf
\end{algorithmic}
\end{algorithm}
6.3. Process Querying

structure of the query feature (line 17). Suppose that \{pqf_1, pqf_2, \ldots, pqf_s\} is the direct parent feature set of a query feature \(qf\) and the matching feature set for \(pqf_i\) \((1 \leq i \leq s)\) is \(PRS_i\); for features in \(PRS_i\), the union of their direct child feature set are computed, denoted as \(RS_i\); for each feature in \(RS_i\), an algorithm is performed to check whether the structure of the feature is matching, i.e., the edges between matching nodes are also matching; if not the feature is removed from \(RS_i\); then the intersection of \(RS_1, RS_2, \ldots, RS_s\), contains all features that are potentially with \(qf\). A PFIndex does not contain advanced features, therefore does not support retrieve matching features for advanced features. Advanced features are left to be checked afterward (see Section 6.3.3).

Figure 6.5 presents a PFIndex with an inverted index (help retrieve node features more efficiently), which is constructed with features of graph 1-5 in Figure 6.2. For example, a query sequence feature, \('Order' → 'Receive'\) is performed through this PFIndex. Firstly, the inverted index helps retrieve node features containing \('Order' ('Receive')\) in their labels. \('Order', is matching with node features, \('Order Goods' and \('Order Goods Online'); \('Receive' is matching with \('Receive Goods' and \('Receive Application. Then, the PFIndex helps retrieve direct child features of matching node features retrieved through the inverted index, which contains all possible matches for the query sequence feature. The direct child features of \('Order Goods' and \('Order Goods Online' are \('Order Goods' → 'Receive Goods', \('Order Goods Online' → 'Receive Goods', and \('Order Goods' → 'Xor-Split'; the direct child features of \('Receive Goods' and \('Receive Application are \('Order Goods' → 'Receive Goods', \('Order Goods Online' → 'Receive Goods', \('Receive Goods' → 'Pay', \('Xor-Split' → 'Receive Goods', \('Receive Goods' → 'Order Goods', and \('Receive Application' → 'Approve'. The common features are \('Order Goods' → 'Receive Goods', \('Order Goods Online' → 'Receive Goods', \('Receive Goods' → 'Pay', and \('Receive Goods' → 'Order Goods'. Finally, the structure is checked. The structure of \('Receive Goods' → 'Order Goods' is not matching with the query. Therefore, the matching features are \('Order Goods' → 'Receive Goods' and \('Order Goods Online' → 'Receive Goods'.

6.3.3 Feature-based Querying

Feature based querying is done by first finding matching graphs for each of the features and subsequently determining whether the matching graphs also match
the query as a whole. For a graph to match the query as a whole, it must meet three requirements. First, it must be a match for all of the basic query features. Second, the mappings that create the matches for each basic query feature, must not contradict each other (i.e.: if a node from the query graph is mapped to a node from the process graph for one feature, it must be mapped to the same node for another feature). Third, the advanced features must be matching with the graph for the given mappings according to Definition 6.12.

Figure 6.5: Example of a PFIndex with an inverted index
More precisely, feature-based querying is defined as follows.

**Definition 6.13** (Feature-based Querying). Given a business process graph $g$, a query graph $q_g$ and a decomposition of the query graph into a set of basic query features $\{f_1, f_2, \ldots, f_n\}$ and a set of advanced query features $\{a f_1, a f_2, \ldots, a f_n\}$ (as defined in Definition 6.9). The business process graph matches the query graph, if and only if for each of the features there exists a corresponding mapping $M_1, M_2, \ldots, M_n$, such that:

- each mapping $M_i$ creates a match of query $f_i$ to $g$ according to Definition 6.12;
- there exists the mapping $M$ such that for each node $n \in N_Q$, for each basic feature $f_i$, if $n$ is a node of $f_i$, $M(n) = M_i(n)$ ($1 \leq i \leq n$);
- for the mapping $M$, each advanced query feature $a f_i$ is matching with the process graph $g$ according to Definition 6.12.

For example, Figure 6.6 shows a query process graph and three process graphs. If we use only nodes and sequences of size 1 as features, then query $e$ has four basic nodes features, $a$, $b$, $c$ and $d$, the basic sequence features, $a \rightarrow b$ and $b \rightarrow c$, and one negative feature, $b \rightarrow d$.

Firstly, these basic query features are queried through indexes, and retrieved matching features are used to check which graphs contain matches for all basic query features. In this example, graph 7 and graph 8 have matching features for basic query features, while graph 6 do not have and therefore is not a match for the query graph.

Secondly, for each graph satisfying the first requirement, whether the same query node maps to the same node in the graph in all the mappings for features is checked. In this example, graph 7 is also not a match. Although it is a match for all of the basic features, the only possible way in which to make both features match, causes a contradiction in the mappings. In particular the node labeled ‘$b$’ from the query graph must be mapped to two different nodes in graph 7 to match all basic features.

Thirdly, for each graph satisfying the first two requirements, whether advanced features are matching with the graph is checked based on the node mappings above. In this example, graph 8 is a match for the query graph, because it does not have an edge between nodes $b$ and $d$. 
Chapter 6. Applying PFIndex to Process Retrieval

Performing process similarity search by comparing process models are not efficient enough, which can take minutes for a collection of thousands process models [30]. While a search engine, e.g., Google, should respond within milliseconds. Therefore, this section presents how to perform process similarity search more efficiently. It compares features of process models to estimate the process similarity, after which only a small number of process models needs to be compared with the search model directly. It provides an index for features, PFIndex, using which only a small number of process features of the collection needs to be compared with the search features. It provides an improved greedy algorithm, outperforming the current fastest algorithm for process similarity search [30], using which the comparisons between the small number of process models with the search model take less execution time.

The process similarity search technique in this thesis works in three steps, as shown in Fig. 6.7. Firstly, an estimation of the similarity of process models to

---

**Figure 6.6:** Example of feature-based querying
the search model is made, based on the ratio of matching features, and models are classified based on their estimated similarity (Section 6.4.1). Secondly, an improved greedy algorithm for determining process similarity is used to efficiently compute similarities between the search model and potentially relevant models (Section 6.4.2). Finally, the models in the collection are ranked according to their (estimated) similarity to the search model (Section 6.4.3).

Figure 6.7: Steps of the Algorithm

Figure 6.8 presents a search process graph and five process graphs as a collection. In the remainder of this section these graphs are used to illustrate the process similarity search technique in this thesis.

6.4.1 Feature-based Similarity Estimation

How to retrieve matching features for a query feature through a PFIndex is explained in Section 6.3.2 in the context of process querying; while feature retrieval through a PFIndex for process similarity is the same as that for process querying, therefore it is not repeated in this section anymore. In this section, we use the fraction of matching features between two business process models to estimate their similarity, as shown in Definition 6.14.

Definition 6.14 (Estimated Business Process Model Similarity). Given a search process graph $G_q$ and another process graph $G$, with feature sets $F_q$ and $F$ derived...
Chapter 6. Applying PFI\textsuperscript{d}ex to Process Retrieval

from $G_q$ and $G$. The estimated business process similarity, denoted $ESim(G_q, G)$ is the number of features in $G_q$ or $G$ that are matched by a feature in the other process graph, divided by the number of all features in $G_q$ and $G$:

$$\frac{|\{f_q \in F_q | \exists f \in F : match(f_q, f)\}| + |\{f \in F | \exists f_q \in F_q : match(f_q, f)\}|}{|F_q| + |F|}$$

Note that we count the number of features in $G_q$ that match a feature in $G$ separately from the number of features in $G$ that match a feature in $G_q$, because the match is not necessarily one-to-one. For example, a label feature ‘Fill-out Request Forms’ can match with label features ‘Fill-out Requester’s Detail’ and ‘Fill-out Request Details’ in the other process graph.

Based on the estimated graph similarity, we can classify graphs as relevant, irrelevant or potentially relevant to a search graph. We do that by defining the minimal estimated similarity that a graph must have to the search graph to be considered relevant and the minimal estimated similarity that a graph must have to be considered potentially relevant. We return relevant graphs directly, check the potentially relevant graphs with expensive similarity search algorithms \cite{30, 34}, and discard irrelevant graphs.

**Definition 6.15 (Graph Relevance Classification).** Given a search process graph $G_q$ and another process graph $G$, we classify $G$ as:

- relevant to $G_q$ if and only if $ESim(G_q, G) \geq \text{ratio}_r$
- potentially relevant to $G_q$ if and only if $\text{ratio}_r > ESim(G_q, G) > \text{ratio}_p$
- irrelevant to $G_q$ if and only if $\text{ratio}_p \geq ESim(G_q, G)$

Where $\text{ratio}_r$ and $\text{ratio}_p$ are parameters that determine when a process graph is considered to be relevant, potentially relevant or irrelevant and can be set as desired, to produce the best results.

6.4.2 The Improved Greedy Algorithm

In Section 6.4.1, we classify process models as relevant, potentially relevant or irrelevant to a given search model. Potentially relevant models still need to be checked by algorithms that can compute exact process similarity. This section explains how to do this. In previous work, a metric is defined to measure process similarity, and algorithms are given to compute the similarity automatically \cite{30}. In
6.4. Process Similarity Search

In this section, we briefly introduce the metric and the currently fastest algorithm to compute the similarity automatically, the greedy algorithm \[30\]. Then we propose three improvements to the greedy algorithm to further improve its performance.

The Greedy Algorithm for Process Similarity Search

The similarity of two business process graphs is defined as a metric based on the graph edit distance as defined, as described in Definition 6.16.

**Definition 6.16 (Graph Similarity).** Let \( G_1 = (N_1, E_1, \lambda_1) \) and \( G_2 = (N_2, E_2, \lambda_2) \) be two graphs. The graph edit distance between two graphs is the minimal number of atomic operations needed to transform \( G_1 \) into \( G_2 \) or vice versa. Atomic operations include inserting, deleting, and substituting nodes and edges. Let \( M : N_1 \rightarrow N_2 \) be a partial injective mapping that maps \( N_1 \) to \( N_2 \). Let \( n_1 \in N_1 \) be a node in \( G_1 \). \( n_1 \) is a substituted node if and only if \( \exists n_2 \in N_2, M(n_1) = n_2 \), and accordingly \( n_2 \) is also a substituted node. A node \( n \in N \) is a skipped node if and only if it is not a substituted node. Let \( n_{11}, n_{12} \in N_1 \) and \( (n_{11}, n_{12}) \in E_1 \) be two nodes and an edge of \( G_1 \). \( (n_{11}, n_{12}) \) is a skipped edge if and only if \( \exists (n_{21}, n_{22}) \in E_2, M(n_{11}) = n_{21} \land M(n_{12}) = n_{22} \). Similarly, we can define the skipped edge in \( G_2 \).

Let \( \text{subn}, \text{skipn} \) and \( \text{skipe} \) be the sets of substituted nodes, skipped (inserted or deleted) nodes, and skipped (inserted or deleted) edges respectively. The fraction of inserted or deleted nodes, denoted \( \text{fskipn} \), the fraction of inserted or deleted edges, denoted \( \text{fskipe} \), and the average distance of substituted nodes, denoted \( \text{fsubn} \), are defined as follows:

\[
\text{fskipn} = \frac{|\text{skipn}|}{|N_1| + |N_2|}, \quad \text{fskipe} = \frac{|\text{skipe}|}{|E_1| + |E_2|}, \quad \text{fsubn} = \frac{2 \cdot \sum_{(n,m) \in M} 1.0 - \text{lsim}(n,m)}{|\text{subn}|}
\]

The partial graph similarity induced by the mapping \( M \), denoted as \( \text{GSim}(G_1, G_2, M) \), is defined as follows:

\[
\text{GSim}(G_1, G_2, M) = 1.0 - \frac{\text{wskipn} \cdot \text{fskipn} + \text{wskipe} \cdot \text{fskipe} + \text{wsubn} \cdot \text{fsubn}}{\text{wskipn} + \text{wskipe} + \text{wsubn}}
\]

where \( \text{wsubn} \), \( \text{wskipn} \) and \( \text{wskipe} \) are the weights that we assign to substituted nodes, skipped nodes, and skipped edges respectively. These parameters can be set as desired to produce the best results.

The graph similarity of two graphs, denoted as \( \text{GSim}(G_1, G_2) \), is the maximal possible similarity induced by a mapping between these graphs.
As an example, consider the search graph and graph 1 in Fig. 6.9. Let the weights $w_{\text{subn}} = 1.0$, $w_{\text{skipn}} = 0.5$ and $w_{\text{skipe}} = 0.5$. Let the mapping $M = \{\text{("Buy Goods," "Buy Goods")}, \text{("Reception of Goods," "Receive Goods")}\}$. Then, the partial graph similarity induced by $M$ for the search graph, $G_q$, and graph 1, $G_1$, can be computed based on Definition 6.16. Note that there are 2 skipped nodes ("Consume Goods" and "Verify Invoice"), 3 skipped edges, and that the label similarity of "Reception of Goods" and "Receive Goods" is 0.62. Consequently, $\text{GSim}(G_q, G_1, M) = 1.0 - \frac{0.5 \cdot 0.33 + 0.5 \cdot 0.6 + 1.0 \cdot 0.19}{0.5 + 0.5 + 1.0} \approx 0.68$. This is also the maximal possible graph similarity induced by any mapping and, hence, this is the graph similarity of the two graphs, i.e., $\text{GSim}(G_q, G_1) = \text{GSim}(G_q, G_1, M)$.

Algorithm 6 describes the greedy algorithm for computing the graph similarity of two process graphs. The algorithm finds the mapping $M$ for which two process graphs have the highest similarity. The algorithm works as follows. Initially, all possible node pairs are added to openpairs (line 2) and no node pair to the mapping $M$ (line 3). Then, in each iteration, $\text{GSim}(G_1, G_2, M \cup \{(n, m)\})$ is computed for all $(n, m) \in \text{openpairs}$ to select the pair that increases the partial graph similarity the most (line 6-7). That pair is added to the mapping $M$ (line 9) and all pairs that contain one of the nodes from that pair are removed from openpairs (line 10), such that each node can be mapped at most once. The algorithm ends when there is no node pair in openpairs that can increase the graph similarity (line 6-7).

We illustrate the algorithm using again the example based on the search graph and graph 1 from Fig. 6.9. Initially, there are 9 (3 times 3) pairs in openpairs. In the first iteration, the pair that increases the similarity most is ("Buy Goods", "Buy Goods"), because it has the highest label similarity. This pair is added to the mapping $M$ and all elements from openpairs that contain one of the nodes
Algorithm 6: Greedy Algorithm \[30\]

**input**: two business process graphs $G_1 = (N_1, E_1, \lambda_1)$, $G_2 = (N_2, E_2, \lambda_2)$

1. **init**
2. $openpairs \leftarrow N_1 \times N_2$;
3. $M \leftarrow \emptyset$;
4. **begin**
5. **while**
6. $\exists (n, m) \in openpairs$, $GSim(G_1, G_2, M \cup \{(n, m)\}) > GSim(G_1, G_2, M)$ \land
7. $\forall (x, y) \in openpairs$,
   $GSim(G_1, G_2, M \cup \{(x, y)\}) > GSim(G_1, G_2, M \cup \{(n, m)\})$
8. **do**
9. $M \leftarrow M \cup \{(n, m)\}$;
10. $openpairs \leftarrow \{(x, y) \in openpairs | x \neq n, y \neq m\}$;
11. **return** $GSim(G_1, G_2, M)$;

“Buy Goods” are removed from, such that there are 4 (2 times 2) pairs left. In the second iteration, (“Reception of Goods”, “Receive Goods”) is chosen, since the pair increases the partial graph similarity most. Then, there is only one (one by one) pair left, but it cannot increase the partial graph similarity and the function ends. In the example, the graph similarity is computed 14 times (9 times in the first iteration, 4 times in the second iteration and 1 time in the third and last iteration).

**Improvements**

Below, we optimize the algorithm, by reducing the number of times that the graph similarity has to be computed and by reducing the complexity of computing the similarity itself. We present three improvements for Algorithm \[6\].

Firstly, we select only the top-k similar node pairs.

We can reduce the number of times that the graph similarity has to be computed, by initially reducing the number of $openpairs$. In Algorithm \[6\] $openpairs$ is assigned $N_1 \times N_2$ initially (line 3) to include all the possible node mappings between the search graph and the graph in the dataset. However, only pairs with high similarity scores are valuable because those can be expected to increase the similarity the most.
The time complexity of the algorithm is directly related to the size of openpairs. Consequently, reducing the size of this set has direct impact on the execution time. Therefore, in this section we aim to reduce the size of openpairs as follows. For each node in the search graph, we find the top k most similar nodes in the graph and only put these node pairs in the openpairs. Definition 6.17 presents the formal definition of the top k most similar nodes.

**Definition 6.17 (Top-K Most Similar Nodes).** Let \( n \) be a node, let \( N \) be a set of nodes, let \( k \) be the number of similar nodes that should be considered for each node, and let \( \text{sim} \) be a function to compare the similarity of two nodes (e.g., \( \text{lsim} \)). The set that contains the top \( k \) similar nodes for the node \( n \), denoted as \( \text{TOP}(k, n, N, \text{sim}) \), is the set that makes the following conditions hold:

- \( \text{TOP}(k, n, N, \text{sim}) \subseteq N \)
- \( |\text{TOP}(k, n, N, \text{sim})| = \min(k, |N|) \)
- \( \forall p \in \text{TOP}(k, n, N, \text{sim}), \not\exists o \in N/\text{TOP}(k, n, N, \text{sim}), \text{such that } \text{sim}(n, o) > \text{sim}(n, p). \)

The parameter \( k \) can be set as desired to get the best results.

For example, consider the search graph and graph 1 from Fig. 6.9. Let \( k = 1 \), let \( n \) be the node “Reception of Goods” in Search graph, let \( N \) be the node set of graph 1, and let \( \text{sim} = \text{lsim} \). Then, \( \text{TOP}(1, n, N, \text{lsim}) = \{\text{“Receive Goods”}\} \).

There is a drawback to computing the top k node pairs by only using label similarity. The graph edit similarity considers both the label similarity and the structural similarity. However, the Top-K heuristic does not take structural information into account and therefore may not result in the optimal node pair. For example, considering the search graph and graph 2 in Fig. 6.9, the labels “Buy Goods” and “Purchase Commodities” are related, but their label similarity is only 0.24. Consequently, the pair (“Buy Goods”, “Purchase Commodities”) may not be put into openpairs when using the Top-K heuristic, even if it may increase the similarity score later on in the execution of the algorithm.

To partly account for this issue, we can also take into account structural information in the Top-K heuristic. We can do this by comparing the sizes of the pre-sets and post-sets of nodes as defined in Definition 6.4. Then, we can compute the similarity of two nodes by considering both the label and role similarities, as described in Definition 6.18.
6.4. Process Similarity Search

Definition 6.18 (Node Similarity). Let \( n, m \in N \) be two nodes. The node similarity is a weighted average value of \( \text{lsim}(n, m) \) and \( \text{rsim}(n, m) \), i.e.,

\[
\text{nsim}(n, m) = \frac{w_l \cdot \text{lsim}(n, m) + w_r \cdot \text{rsim}(n, m)}{w_l + w_r}
\]

where \( w_l \) and \( w_r \) are parameters that can be set as desired to produce the best results.

For example, let \( w_l = 1.0 \) and \( w_r = 0.5 \). Considering the nodes “Buy Goods” and “Purchase Commodities” in search graph and graph 2 of Fig. 6.9, their node similarity is \( 1.0 \cdot 0.24 + 0.5 \cdot 0.67 \approx 0.38 \). Consequently, for \( k = 1 \) (“Buy Goods”, “Purchase Commodities”) would be put in openpairs, while, if we had used label similarity instead of the node similarity, for the node “Buy Goods” in Search graph, the only node pair in the openpairs would have been (“Buy Goods”, “Get Commodities”).

Secondly, we incrementally compute the graph similarity.

We can reduce the computation time of the graph similarity, by computing it incrementally instead of anew in each iteration. In Algorithm 6, when we add a new node pair \((n, m)\) into the mapping \(M\), we need to re-compute the partial graph similarity according to the new mapping \(M \cup \{(n, m)\}\), i.e., \(\text{GSim}(G_1, G_2, M \cup \{(n, m)\})\) (line 7 and 8). However, \(\text{GSim}(G_1, G_2, M \cup \{(n, m)\})\) is related to \(\text{GSim}(G_1, G_2, M)\), so we should compute it incrementally. Therefore, this section investigates the definition of graph similarity and deduces an incremental way to compute the partial graph similarity.

From Definition 6.16, we know that the partial graph similarity is related to three fractions, i.e., \(f_{\text{skipn}}, f_{\text{skipe}}\) and \(f_{\text{subsn}}\). These fractions change when \(M\) changes. Let us see how these fractions change one by one after putting a node pair to \(M\).

First, no matter which node pairs are in \(M\) and which new node pair is put into \(M\), the size of \(\text{skipn}\) always reduces by two, because two nodes are matched and removed from the skipped node set. Thus, we can compute the increment of \(f_{\text{skipn}}\), as defined in Definition 6.19.

Definition 6.19 (Skipped-node Fraction Increment). Let \(G_1 = (N_1, E_1, \lambda_1)\) and \(G_2 = (N_2, E_2, \lambda_2)\) be two graphs. Let \(M\) be a partial injective mapping that maps \(N_1\) to \(N_2\). Let \((n, m)\) be a node pair in openpairs. After putting \((n, m)\) into \(M\), the increment of \(f_{\text{skipn}}\), denoted as \(\Delta[f_{\text{skipn}}]\), is defined as follows:

\[
\Delta[f_{\text{skipn}}] = \frac{-2}{|N_1| + |N_2|}
\]
Second, the size reduction of $\text{skipe}$ is related to the new node pair $(n, m)$ and the mapping $M$. We can compute this by only considering edge pairs that are related to $(n, m)$, instead of all possible edge pairs. The size reduction is equal to the size of the intersection of $\bullet n \times \bullet m$ and $M$ and the size of the intersection of $n \bullet \times m \bullet$ and $M$.

**Definition 6.20** (Skipped-edge Increment). Let $G_1 = (N_1, E_1, \lambda_1)$ and $G_2 = (N_2, E_2, \lambda_2)$ be two graphs. Let $M$ be a partial injective mapping that maps $N_1$ to $N_2$. Let $(n, m)$ be a node pair in $\text{openpairs}$. After putting $(n, m)$ into $M$, the increment of $|\text{skipe}|$, denoted as $\Delta_{|\text{skipe}|}$, is defined as follows:

$$\Delta_{|\text{skipe}|} = -2 \cdot (|\{(x, y) \in M | x \in \bullet n \land y \in \bullet m\}| + |\{(x, y) \in M | x \in n \bullet \land y \in m \bullet\}|)$$

The increment of $f_{\text{skipe}}$ is defined in Definition 6.21.

**Definition 6.21** (Skipped-edge Fraction Increment). Let $G_1 = (N_1, E_1, \lambda_1)$ and $G_2 = (N_2, E_2, \lambda_2)$ be two graphs. Let $M$ be a partial injective mapping that maps $N_1$ to $N_2$. Let $(n, m)$ be a node pair in $\text{openpairs}$. After putting $(n, m)$ into $M$, the increment of $f_{\text{skipe}}$, denoted as $\Delta_{f_{\text{skipe}}}$, is defined as follows:

$$\Delta_{f_{\text{skipe}}} = \frac{\Delta_{|\text{skipe}|}}{|E_1| + |E_2|}$$

Third, contrary to $\text{skipn}$, the size of $\text{subn}$ increases by two after putting $(n, m)$ into $M$. The increment of $f_{\text{subn}}$ also involves the label similarities of $(n, m)$ and pairs in $M$, as defined in Definition 6.22.

**Definition 6.22** (Substituted-node Fraction Increment). Let $G_1 = (N_1, E_1, \lambda_1)$ and $G_2 = (N_2, E_2, \lambda_2)$ be two graphs. Let $M$ be a partial injective mapping that maps $N_1$ to $N_2$. Let $(n, m)$ be a node pair in $\text{openpairs}$. After putting $(n, m)$ into $M$, the increment of $f_{\text{subn}}$, denoted as $\Delta_{f_{\text{subn}}}$, is defined as follows:

$$\Delta_{f_{\text{subn}}} = \frac{\Sigma_{(x,y) \in M} l_{\text{sim}}(x,y)}{|\text{subn}|} - \frac{\Sigma_{(x,y) \in M \cup \{(n,m)\}} l_{\text{sim}}(x,y)}{|\text{subn}| + 2}$$

where $|\text{subn}| = 2 \cdot |M|$. From the analysis above, we can derive that after putting $(n, m)$ into $M$, the graph similarity increment can be computed by performing only two computations, those of $l_{\text{sim}}(n,m)$ and $\Delta_{|\text{skipe}|}$. The other components of the computation are either constants that can be computed before executing the algorithm, or functions of
6.4. Process Similarity Search

$M$ that only have to be computed once each time $M$ changes. Proposition 6.1 shows how the graph similarity increment can be computed as a function of $\text{lsim}(n,m)$ and $\Delta_{|\text{skip}|}$, two constants $c_1$ and $c_2$ and two functions of $M$: $\varphi_1(M)$ and $\varphi_1(M)$.

**Proposition 6.1 (Graph Similarity Increment).** Let $G_1 = (N_1, E_1, \lambda_1)$ and $G_2 = (N_2, E_2, \lambda_2)$ be two graphs. Let $M$ be a partial injective mapping that maps $N_1$ to $N_2$. Let $(n,m)$ be a node pair in openpairs. After putting $(n,m)$ into $M$, the graph similarity increment, denoted as $\Delta$, is defined as follows:

$$\Delta = \varphi_1(M) \cdot \text{lsim}(n,m) + c_1 \cdot \Delta_{|\text{skip}|} + \varphi_2(M) + c_2,$$

where:

$$c_1 = \frac{-\text{wskipe}}{(\text{wskipn} + \text{wskipe} + \text{wsubn}) \cdot \left(|E_1| + |E_2|\right)},$$
$$c_2 = \frac{2 \cdot \text{wskipn}}{(\text{wskipn} + \text{wskipe} + \text{wsubn}) \cdot \left(|N_1| + |N_2|\right)},$$
$$\varphi_1(M) = \frac{\text{wsubn}}{(\text{wskipn} + \text{wskipe} + \text{wsubn}) \cdot \left(|M| + 1\right)}, \text{ and }$$
$$\varphi_2(M) = \frac{-\text{wsubn} \cdot \Sigma_{(x,y) \in A} \text{lsim}(x,y)}{(\text{wskipn} + \text{wskipe} + \text{wsubn}) \cdot \left(|M| + 1\right) \cdot |M|}.$$

We prove Proposition 6.1 as follows.

**Proof.** The graph similarity increment is equal to the graph similarity after adding $(n,m)$, i.e.: $\Delta = \text{GSim}(G_1, G_2, M \cup \{(n,m)\}) - \text{GSim}(G_1, G_2, M)$. We can rewrite this as follows.

$$\Delta = \text{GSim}(G_1, G_2, M \cup \{(n,m)\}) - \text{GSim}(G_1, G_2, M)$$
$$= (1.0 - \frac{\text{wskipn} \cdot \text{fskipn}' + \text{wskipe} \cdot \text{fskipe}' + \text{wsubn} \cdot \text{fsubn}'}{\text{wskipn} \cdot \text{wskipe} \cdot \text{wsubn}}) -$$
$$\left(1.0 - \frac{\text{wskipn} \cdot \text{fskipn} \cdot \text{fskipe} \cdot \text{fsubn}}{\text{wskipn} \cdot \text{wskipe} \cdot \text{wsubn}} \right)$$
$$= \frac{-\text{wskipe} \cdot \Delta_{|\text{skip}|} - \text{wskipn} \cdot \Delta_{|\text{skip}|} - \text{wsubn} \cdot \Delta_{|\text{sub}|}}{\text{wskipn} \cdot \text{wskipe} \cdot \text{wsubn}}$$
$$= \frac{1}{\text{wskipn} \cdot \text{wskipe} \cdot \text{wsubn}} \cdot \left(1 - \frac{\text{wskipe}}{|N_1| + |N_2|} - \frac{\text{w提速}}{|E_1| + |E_2|} \cdot \frac{\Delta_{|\text{skip}|}}{|M| + 1} \right)$$
$$= \varphi_1(M) \cdot \text{lsim}(n,m) + c_1 \cdot \Delta_{|\text{skip}|} + \varphi_2(M) + c_2$$

From Proposition 6.1 we can see that only $\text{lsim}(n,m)$ and $\Delta_{|\text{skip}|}$ are related to the new pair $(n,m)$. We already know how to compute them as described in Definitions 6.2 and 6.20. Therefore, we can compute the graph similarity incrementally. For example, considering search graph and graph 1 from Fig. 6.9. Let the weights wsubn = 1.0, wskipn = 0.5 and wskipe = 0.5. Let
the mapping \( M = \{ ( "Buy\ Goods", "Buy\ Goods") \} \). Then, after putting ("Reception of Goods","Receive Goods") into \( M \), the graph similarity increment is
\[
\frac{1}{2.0} \cdot \left( \frac{1.0 \cdot 0.63}{2} - \frac{0.5 \cdot (-2)}{5} - \frac{1.0 \cdot 1.0}{2} + \frac{2 \cdot 0.5}{6} \right) \approx 0.1.
\]

Thirdly, we pre-select most similar node pairs.

We can reduce the number of times the graph similarity must be computed, by ‘predicting’ the pair in the mapping that would increase the similarity the most. In Algorithm 6 to decide which node pair to add into the mapping \( M \) next, we need to compute \( GSim(G_1, G_2, M \cup \{(n,m)\}) \) for all the \((n,m)\) in openpairs and find the maximal value (lines 7 and 8). Proposition 6.1 discloses that the graph similarity increment is related to two variables: \( lsim(n,m) \) and \( \Delta_{\{\text{skip}\}} \) only. Consequently, if we can ‘predict’ the value of those two variables, we can predict the value of the overall similarity increase.

This section first proposes an efficient manner to compute the value of \( \Delta_{\{\text{skip}\}} \). Then, it uses the values of \( lsim(n,m) \) and \( \Delta_{\{\text{skip}\}} \) to pre-select a few candidate pairs that potentially have the largest similarity increment. Last, it finds the node pair with the maximal graph similarity increment from candidate pairs. By doing this, only the graph similarity increments of these candidate pairs, instead of all the pairs in openpairs, need to be computed and compared to find the pair with the maximal graph similarity increment.

We can efficiently compute the value of \( \Delta_{\{\text{skip}\}} \) as follows. During iterations, given a pair \((n,m)\), \( lsim(n,m) \) is constant in spite of the changes of the mapping \( M \). However, \( \Delta_{\{\text{skip}\}} \) is related to both \((n,m)\) and \( M \). In each iteration, we need to know the \( \Delta_{\{\text{skip}\}} \) values for all the pairs in openpairs before the pre-selection. We can compute the \( \Delta_{\{\text{skip}\}} \) values based on Definition 6.20 in each iteration, but it is time consuming because of the consideration of all pairs in openpairs. Instead, we build a cache to store the \( \Delta_{\{\text{skip}\}} \) values for all the pairs in openpairs. Initially, all the \( \Delta_{\{\text{skip}\}} \) values are 0, because there is no node pair in \( M \). When a node pair \((o,p)\) is added to \( M \), we only need to update the \( \Delta_{\{\text{skip}\}} \) values for \((n,m)\) \( \in \) openpair that makes \( n \in o \land m \in p \) or \( n \in o \lor m \in p \) hold. Proposition 6.2 presents the rule to update the \( \Delta_{\{\text{skip}\}} \) values.

**Proposition 6.2** (Difference of Skipped-edge Increment). Let \( G_1 = (N_1, E_1, \lambda_1) \) and \( G_2 = (N_2, E_2, \lambda_2) \) be two process graphs as defined in Definition 5.1. Let \( M \) be a partial injective mapping that maps \( N_1 \) to \( N_2 \). Let \((o,p)\) and \((n,m)\) be two node pairs in openpairs \((o \neq n \land p \neq m)\). After putting \((o,p)\) into \( M \), the difference
of the $\Delta_{\text{skip}}$ value for $(n,m)$ is defined as follows:

$$-2 \cdot (|\{(x,y) \in \{(o,p)\}|n \in \bullet x \land m \in \bullet y| + |\{(x,y) \in \{(o,p)\}|n \in x \land m \in \bullet y\}|).$$

We prove Proposition 6.2 as follows.

**Proof.** Let $M' = M \cup \{(o,p)\}.

$$\Delta'_{\text{skip}} - \Delta_{\text{skip}} = -2 \cdot (|\{(x,y) \in M'|x \in n \land y \in m\}| + |\{(x,y) \in M'|x \in n \land y \in m\}| - |\{(x,y) \in M|x \in n \land y \in m\}| - |\{(x,y) \in M|x \in n \land y \in m\}|)$$

$$= -2 \cdot (|\{(x,y) \in \{(o,p)\}|x \in n \land y \in m\}| + |\{(x,y) \in \{(o,p)\}|x \in n \land y \in m\}| + |\{(x,y) \in \{(o,p)\}|x \in n \land y \in m\}|).$$

As an example, consider search graph and graph 1 from Fig. 6.9. Let the mapping $M = \emptyset$. Then, after putting (“Buy Goods”, “Buy Goods”) into $M$, only the $\Delta_{\text{skip}}$ values for (“Reception of Goods”, “Receive Goods”) and (“Reception of Goods”, “Verify Invoice”) need to be modified to -2.

By now, we already know the values of $l_{\text{sim}}(n,m)$ and $\Delta_{\text{skip}}$. Then, let us see how to use them to pre-select candidate pairs. The value range of $\Delta_{\text{skip}}$ is typically limited. For example, in the validation dataset of this article, it can only be 0, -2, -4, or -6 (the values are always even, because only two edges can match each other at one time). Therefore, we can first consider $\Delta_{\text{skip}}$ and then $l_{\text{sim}}(n,m)$. For each possible value of $\Delta_{\text{skip}}$, the node pair $(n,m)$ with the maximal $l_{\text{sim}}(n,m)$ value is selected as a candidate pair (see Definition 6.1). To quickly get the pair $(n,m)$ with maximal label similarity, we can sort the node pairs in openpair descendingly with respect to their label similarities in advance. We get a few candidate pairs, of which the one with the maximal graph similarity increment is the pair we are looking for.

Next, we can pre-select the potential node pairs to be put into $M$. For example, search graph and graph 1 in Fig. 6.9 are considered. When $M = \{\{\text{“Buy Goods”}, \text{“Buy Goods”}\}\}$, the distinct values for $\Delta_{\text{skip}}$ are -2 or 0. The node pairs (“Reception of Goods”, “Receive Goods”) and (“Consume Goods”, “Receive
Chapter 6. Applying PFIndex to Process Retrieval

Goods”) are selected as candidate pairs respectively for each $\Delta_{\text{skip}}$ value. Finally, (“Reception of Goods”, “Receive Goods”) is put into $M$, because it provides a higher graph similarity increment.

There are algorithms solving multiple items with the highest overall values, e.g., the well-known threshold algorithm (TA) \[40\]. These algorithms are also applicable for node pair pre-selection (two items in this case, $\Delta_{\text{skip}}$ and $l_{\text{sim}}(n, m)$). In this section, we present a simple but effective algorithm for the sake of explanation.

The Improved Greedy Algorithm for Process Similarity Search

This sub-section integrates the improvements proposed above into the Greedy Algorithm (Algorithm 6) and presents the improved algorithm after integration, as shown in Algorithm 7.

Initially, instead of considering all the possible node pairs, only the top $k$ most similar nodes are considered for each search node (see line 8). These node pairs are sorted with respect to their label similarities. The mapping is empty at first (see line 9). Three more variables are defined (see lines 10–12): $\text{result}$, $\text{skipedgecache}$ and $\text{candidatepairs}$. $\text{result}$ is the partial graph similarity for the current mapping $M$. $\text{skipedgecache}$ is a list that records the numbers of potentially matched edges for each pair in $\text{openpairlist}$. $\text{candidatepairs}$ is a mapping that, for each possible value in $\text{skipedgecache}$, records the node pair with maximal node similarity.

In each iteration, the node pair with the maximal graph similarity increment in $\text{candidatepairs}$ is added to the mapping; the variables are adapted according to the current state after that (see lines 13–24). The function ends when there is no more node pair for which the graph similarity increases.

Fig. 6.10 shows an example in which the similarity of the search graph and graph 1 from Fig. 6.9 are computed according to the improved algorithm. Let $k = 2$, $w_{\text{skipn}} = w_{\text{skipe}} = w_l = 0.5$, and $w_{\text{subn}} = w_r = 1.0$. In the figure the nodes are identified by the first letters of the words in their labels. The figure shows the values for the variables of the algorithm for three iterations. Initially, the top-2 most similar nodes for each node in the search model are determined. Based on that information, the $\text{openpairlist}$ is constructed. Initially, none of the pairs in the $\text{openpairlist}$ will reduce the number of skipped edges. Consequently, the $\text{skipedgecache}$ contains only 0s. The only pair in $\text{candidatepairs}$ is (“Buy Goods”, “Buy Goods”), which is consequently put into $M$ in the first iteration. As
Algorithm 7: Improved Greedy Algorithm

input: two business process graphs $G_1 = (N_1, E_1, \lambda_1), G_2 = (N_2, E_2, \lambda_2)$

function $\text{sortedTopkPairs}(N_1, N_2)$;

begin

    foreach $n_1 \in N_1$ do
        select the top k $n_2 \in N_2$ with respect to $nsim(n_1, n_2)$;
        put the top k $(n_1, n_2)$ pairs in openpairlist;
    
    return sorted openpairlist with respect to $lsim(n_1, n_2)$ in descending order;

init

    openpairlist $\leftarrow$ sortedTopkPairs($N_1, N_2$);
    $M \leftarrow \emptyset$;
    $result \leftarrow 0$;
    skipedgecache $\leftarrow [0, 0, \cdots, 0]$;
    candidatepairs $\leftarrow [(0, \text{Head(openpairlist)})]$;

begin

    while $\exists (n, m) \in \text{candidatepairs}, \Delta > 0 \land \exists (x, y) \in \text{candidatepairs}, \Delta' > \Delta$ do
        $M \leftarrow M \cup \{(n, m)\}$;
        $result + = \Delta$;
        foreach $(x, y) \in \text{openpairlist} \land (x = n \lor y = m)$ do
            remove $(x, y)$ from openpairlist;
            remove the according item for $(x, y)$ from skipedgecache;
        foreach $(x, y) \in \text{openpairlist} \land ((x \in \bullet n \land y \in \bullet m) \lor (x \in n \bullet \land y \in m\bullet))$ do
            update skipedgecache; // skipedge$− = 2$ or skipedge$− = 4$
        foreach distinct value in skipedgecache do
            update candidatepairs; //store maximal $lsim(n, m)$
    
    return $result$;

end

a result, the openpairlist, skipedgecache and candidatepairs variables are updated. There are now two distinct values (-2 and 0) in skipedgecache, and two pairs in
Chapter 6. Applying PFIndex to Process Retrieval

candidatepairs. (“Reception of Goods”, “Receive Goods”) provides higher graph similarity increase and is chosen to be in \( M \) in the second iteration. Then, there is only one pair left in candidatepairs, but it cannot increase the partial graph similarity and the algorithm ends.

During the iterations, only four partial graph similarities induced by different mappings are computed, and only two partial graph similarities are compared to select the pair with the maximal graph similarity increment. Some additional computation is required in Algorithm 7 as compared to the original algorithm (e.g., ranking node pairs with respect to the label similarity). However, the improved algorithm is much less time consuming, as will be shown in the evaluation results in Chapter 8.

6.4.3 Ranking

Using the similarity estimation metric \( E_{Sim} \) from Section 6.4.1 and the similarity measurement metric \( G_{Sim} \) from Section 6.4.2, we can rank the models in a collection in the order of their similarity to a search model.

Given a search business process model and a set of business process models, we classify the set of business process models as ‘relevant’, ‘potentially relevant’ or ‘irrelevant’, according to Definition 6.15. We only rank the models in the ‘relevant’ and the ‘potentially relevant’ sets, by first presenting the models in the ‘relevant’ set, in the order of their estimated similarity \( E_{Sim} \) to the search model, and then
6.4. Process Similarity Search

presenting the models in the ‘potentially relevant’ set in the order of their similarity GSim to the search model. Ranking models in a set, results in a sequence that is ordered in descending order of similarity score (most similar item first). Sequences can be concatenated to produce a complete search result. Given two sequences $L$ and $M$, their concatenation, denoted $L++M$, is the sequence in which the elements from $L$ are put in front of the elements from $M$ operand. We only consider the potentially relevant models that are sufficiently similar to the search model (i.e. we only consider the models $G$, for which $\text{GSim}(G, G_q) > \text{cutoff}$, where cutoff is a parameter). More precisely, the ranking is defined as follows.

**Definition 6.23 (Ranking).** Let $G_q$ be a search graph and let $G_s$ be a set of graphs. Furthermore, let cutoff be a parameter that determines the minimum similarity score. The ranking of the graphs from $G_s$ according to their similarity to $G_q$ is a mathematical sequence $G_r++G_p$, where:

- $G_r$ is the sequence that consists of all models from $G_s$ that are relevant to $G_q$, such that for each $G_{r_i}, G_{r_j}$ from $G_r$ holds: if $i < j$ then $\text{ESim}(G_{r_i}, G_q) \geq \text{ESim}(G_{r_j}, G_q)$;
- $G_p$ is the sequence that consists of all models $G$ from $G_s$ that are potentially relevant to $G_q$ and for which $\text{GSim}(G, G_q) > \text{cutoff}$, such that for each $G_{p_i}, G_{p_j}$ from $G_p$ holds: if $i < j$ then $\text{GSim}(G_{p_i}, G_q) \geq \text{GSim}(G_{p_j}, G_q)$.

The improvement in the time complexity when using the similarity estimation step, can be characterized as follows. Let $k$ be the total number of process models in a collection and $n$ be the average number of nodes in a process model. If node features are used for similarity estimation, the similarity estimation searches the most similar node in a tree-based index, for each node in the search model. There are $k \cdot n$ nodes in the tree at most, when all nodes in the process model collection are distinct from each other. Therefore, the time complexity of the similarity estimation step (using the node features only) has an upper bound of $O(n \cdot \log(k \cdot n))$. The time complexity of the greedy algorithm for process similarity search is $O(n^3)$ \[30\]. Therefore, the improvement in time complexity is characterized as: $O(p \cdot n^3 - n \cdot \log(k \cdot n))$, where $p$ is the fraction of models that can directly be classified as relevant or irrelevant, after the similarity estimation step.
Chapter 7

Implementation

7.1 Introduction

Chapter 6 presents the techniques for process retrieval (process querying and similarity search) using the process feature index (PFIndex). This chapter describes the architecture that we propose for implementing the techniques in Chapter 6. The architecture is based on the more general architecture for business process model repositories in Chapter 2 and focuses on the process retrieval aspect. As such, it provides a more detailed design of a single aspect of the architecture for business process model repositories. As a proof of concept, we implemented a web-based prototype of the architecture and the techniques. \(^1\)

The remainder of the chapter is organized as follows. Section 7.2 presents the general layered architecture of the tool in terms of a component diagram. Section 7.3 makes the architecture more concrete, by presenting details of the interfaces of the components in terms of class diagrams. Section 7.4 presents the prototype that implements the architecture.

\(^1\)Access the prototype at: http://is.tm.tue.nl/research/apromore.html. Please take Firefox or Google Chrome as your web browser, since IE does not support the script we use.
7.2 Component Diagram

Fig. 7.1 presents the general architecture of the tool, which is based on the reference architecture presented in Chapter 2. However, where the reference architecture presents a general architecture that contains all functions that can be implemented by a business process model repository at a high level of abstraction, this thesis presents a detailed architecture for the retrieval function only.

---

**Figure 7.1:** Architecture of the Tool

The architecture consists of four layers: the presentation layer, the process repository management layer, the DBMS layer, and the storage layer. The presentation
7.3. Class Diagrams

Fig. 7.2 shows the architecture in more detail with class diagrams.

The process repository management layer consists of four components. The process graph component provides two operations: “convertGraph” for transforming a given process model into a process graph (Definition 5.1) and “getFeatures” for deriving the features of a given process graph (Definition 5.2 and 6.9). The operation for transforming a process model into a process graph can be overloaded to enable the conversion of multiple process modeling notations. Our prototype supports both the EPC and the BPMN notations. The index management component provides operations for constructing and managing the indexes. It provides an operation “constructIndex” that inserts a collection of process graphs into an empty PFIndex (Algorithm 1); it provides operations “insertGraph”, “updateGraph”, and “deleteGraph” that inserts, updates and deletes a process graph in a PFIndex (Algorithms 2, 4, and 3). The component also provides operations for retrieving process features through the indexes. It provides two operations “queryFeatures”
Chapter 7. Implementation

The external process model includes an `readAll()` method that returns a list of `ProcessModel` objects. The internal process model has methods `readGraph(in graph : ProcessGraph) : void` and `createGraph(in graph : ProcessGraph) : void`, which are used to manipulate process graphs. The tool also provides an interface for retrieval and similarity search, which includes methods like `queryFeatures(in query : ProcessGraph, out results : ProcessGraph)` and `getESim(in fs : Feature[], in mfs : Feature[], in graph : ProcessGraph) : float`.

The figure illustrates the architecture of the tool in detail, showing the relationships between the process repository, management layer, storage layer, and internal process model. The tool processes graphs using features like `tree`, `node`, and `edge`, and implements operations such as `createIndex(out i : PFIndex) : void`, `updateIndex(inout i : PFIndex) : void`, and `readIndex(out i : PFIndex) : void` for index management.

```
Figure 7.2: Architecture of the Tool in Detail
```

and “searchFeatures” that returns, given a basic feature, the matching features in the context of querying and similarity search respectively (Algorithm 5); it provides an operation “check” for process querying that checks whether in a process graph matched features (returned by “queryFeatures”), contradict with each other.
and checks whether the advanced features match with the graph (Definition 6.13); and it provides an operation “getESim” for process similarity search that computes the estimated similarity of two graphs, based on their matched features (returned by “searchFeatures”) (Definition 6.14). The greedy algorithm component provides only a single operation, “getGSim”, for determining process graph similarity (Algorithm 7). The retrieval component provides two operations. Operation “query” for querying the models that satisfy a given model is supported by operations “queryFeatures” and “check” of the index management component (Section 6.3). Operation “search” for searching the models that are similar to a given model is supported by operations “searchFeatures” and “getGSim” of the index management component and an operation “getGSim” of the greedy algorithm component (Section 6.4).

The storage layer consists of three components: the indexing component and the internal and external process model component. The indexing component is the core of our design. It stores features and an index based on features. As examples, Fig. 7.2 contains two types of features. However, sub-classes of “Feature” can be created as desired to also store other types of features. “NodeFeature” stores a label and a number of input and output edges; “Seq1Feature” stores sequences of size one. The class diagram describes the index, “PFIndex”, which stores hierarchical relations between features. More precisely, it stores which feature is a (direct) parent of which other features. The index also stores the relation between features and the business process graphs in which they are contained. The internal process model component stores the business process models in the format that is used in the repository for efficient computation, which is the (query) process graph in this case. The external process model stores the business process models in their original format. Process models are described in the “ProcessModel” class, which has several subclasses, indicating that process models can be described in different notations, e.g., EPC and BPMN. The class can be extended as desired to store other types of models. In order for those models to work in the repository, the process repository management layer must contain functions to convert them to business process graphs. Note that process models, the corresponding process graphs and features of those process graphs are related via the “processId” that must be unique for a given process model.

Fig. 7.2 describes the most important components in detail. We excluded details
about the other components, because they are not essential to understand the design and because they would differ in different repositories, for example, to cater for different GUI requirements or to include business process models in different notations. For the same reason, not all operations that are made available by the repository are shown. For example, the external process model storage component only provides an operation to read all models, but obviously also operations should be provided to create, read, update and delete singular models. These operations, however, are not essential to understanding the design.

7.4 Prototype

As a proof of concept, we implemented a web-based prototype of the architecture in Java with MySQL as the DBMS. Process models are externally stored in XML files and internally stored in MySQL. At this moment, the index is constructed and in-memory, instead of in the storage layer. We implemented it like this in our prototype because this is merely a proof of concept. In practice, the index should also stored in the repository. Fig. 7.3 presents a screenshot of the tool, displaying its main functionality.

![Figure 7.3: A Screenshot of the Tool](image-url)
Chapter 8

Evaluation

8.1 Introduction

Chapter 7 presents a prototype that implements the process retrieval techniques proposed in this part. This chapter presents experiments to evaluate the techniques implemented in the prototype in this thesis in terms of efficiency, quality, and scalability. The efficiency is measured by the average execution time; the quality is measured by the R-Precision (explained latter in this chapter) of retrieving results; the scalability is measured by comparing the efficiency after enlarging the size of the collection by 10 times with the original efficiency.

Two types of business process model collections were used in the experiments, two real-life collections and two synthetic collections. One real-life collection consists of 604 SAP reference models and the other consists of 10 process models from a manufacturing company. One of the synthetic collections consists of 604 synthetic models and the other consists of 6040 synthetic process models. A SAP reference process model on average contains 20.7 nodes and 20.5 edges; a manufacturing process models on average contains 20.3 nodes and 20.1 edges; a synthetic process model on average contains 20.3 nodes and 24.1 edges.

For process querying, two types of experiments were designed. In the first one, we used the collection of SAP reference models to evaluate the efficiency of the technique; in the last one, we generated 6040 synthetic process models to evaluate the scalability of the technique. The algorithms to generate process models we
propose are also explained in this chapter. We did not evaluate the quality of retrieving results for process querying, because process querying only considers exact matches, therefore its R-Precision should be 1. For process similarity search, three types of experiments were designed. In the first one, we used the collection of SAP reference models to evaluate both the efficiency of the technique and the quality of retrieving results; in the second one, we used the manufacturing process models as search models and SAP reference models as a collection to evaluate the quality of searching with heterogeneous models; in the last one, we again used the synthetic models to evaluate the scalability of the technique.

All the experiments were run on a laptop with an Intel Core2 Duo T7500 CPU (2.2GHz, 800MHz FSB, 4MB L2 cache), 4 GB DDR2 memory, the Windows Vista operating system and the SUN Java Virtual Machine version 1.6. We choose a laptop to run the experiments because in the real use cases, the techniques should support users with retrieving process models stored in their laptops or computers.

The remainder of the chapter is organized as follows. Section 8.2 presents a synthetic process model generator. Section 8.3 evaluates the proposed process querying technique. Section 8.4 evaluates the proposed process similarity search technique. Section 8.5 concludes the chapter.

### 8.2 Synthetic Process Model Generator

To evaluate the scalability of the process retrieval techniques, the size of the collection of SAP reference models is not big enough. Therefore, this section presents a synthetic process model generator to generate a large number of synthetic process models. Firstly, synthetic labels are generated by decomposing original labels into words, and then recomposing words to form synthetic labels based on the probability of word occurrence in the original labels. Then synthetic process models are generated by decomposing original process models into features (Definition 5.2), and then recomposing features to form synthetic process models based on the probability of feature occurrence in the original collection. The labels of the features are also replaced by synthetic labels. We generate synthetic collection in this way to remain the characteristics of the original collection in terms of labels and structures for the purpose of evaluating the proposed techniques.

It firstly explains the properties (of labels and structures) of process models in
8.2. Synthetic Process Model Generator

a collection that are relevant to the generation of synthetic models in the context of the process similarity search and querying techniques as we defined in Chapter 5 and 6. Secondly, it presents an algorithm to generate a collection of synthetic process models based on these properties.

8.2.1 Properties of Business Process Model Collections

This section presents properties that are discovered from business process model collections and that are relevant to evaluate the process similarity search and querying techniques defined in Chapter 5 and 6. The two most common aspects of business process models, as shown in Table 3.1 of Chapter 3 are considered, i.e., activity and control flow. For activities, properties of node (label) features are discovered. For control flows, properties of structural features of process models are discovered.

Label Properties

Labels consist of words and words in labels of a given process model collection are composed to form synthetic labels in this section. To generate synthetic labels with similar properties as labels in a collection, we consider two types of label properties regarding words, i.e., the occurrence of a word and the co-occurrence of two words. The former one indicates the frequency and probability of a word appearing in a synthetic label and the later one indicates the frequency and probability of two words appearing in a synthetic label or labels of two connected nodes.

Lower case versions of the words are used. Stop-words, e.g., ‘a’, ‘an’, ‘the’, ‘one’, . . . , and gateway labels, e.g., ‘and-split’, are not considered. Definition 8.1 presents the word set of a process model collection.

Definition 8.1 (Word Set, Label Size). Let \( \mathcal{D} \) be a collection of process graphs with disjoint sets of nodes and let \( \mathcal{L} \) be the label set. The function \( \omega(l) \) maps a label \( l \) to the set of words that appear in \( l \).

The word set \( \mathcal{W} \) of the collection \( \mathcal{D} \) is the set of words appear in \( \mathcal{L} \). Formally, \( \mathcal{W} = \{ w \mid w \in \omega(l) \land l \in \mathcal{L} \} \).

The size of a label \( l \) is the number of words in \( l \), i.e., \( |\omega(l)| \).

For example, the word set of the collection in Figure 8.1 is \{order, goods, receive, online, pay, application, approve\}. 
To generate a label, it is necessary to know which word is in the label. Therefore, the probability of word occurrence is required. For a word in the word set of a collection, the frequency and probability of its occurrence are defined in as follows.

**Definition 8.2 (Frequency of Word Occurrence, Probability of Word Occurrence).**

Let $D$ be a collection of process graphs with disjoint sets of nodes, let $N$ be the node set of $D$, and $W$ be the word set of $D$.

The frequency of the occurrence of a word $w$, denoted as $FWO(w)$, is the number of nodes in the collection that contain the word $w$ in their labels. Formally, $FWO(w) = |\{n \in N | w \in \omega(\lambda(n))\}|$.

The probability of the occurrence of a word $w$ is the frequency of the occurrence of $w$ divided by the frequency of the occurrence of all words. Formally, $PWO(w) = \frac{FWO(w)}{\sum_{w_1 \in W} FWO(w_1)}$.

For example, in Figure 8.1, the frequency of the word occurrence of ‘goods’ is 8, and its probability is $8/22=0.36$.

To generate a label, it is also necessary to know which words can occur in the same label or labels of two connected nodes. Therefore, three types of word co-occurrence are considered as defined in Definition 8.3, i.e., word co-occurrence, pre-word co-occurrence, and post-word co-occurrences. Gateway nodes are not considered for pre-word (post-word) co-occurrence. For a node $n$, if a node $n_1 \in \bullet n$ ($n \bullet$) is a gateway node, nodes in $\bullet n_1$ ($n_1 \bullet$) of the gateway nodes are considered instead of $n_1$. For example, in graph 4 of Figure 8.1 the post-word co-occurrence for words in the node ‘order goods’, nodes ‘receive goods’ and ‘pay’ are considered instead of the gateway node ‘and-split’.

Given a word $w$ in a node $n$, we need to know which word can appear together with $w$ in the same node $n$; which word can appear in a node in the pre-set of $n$; which word can appear in a node in the post-set of $n$. The frequencies of three
8.2. Synthetic Process Model Generator

types of word co-occurrence are defined as follows.

**Definition 8.3** (Frequency of Word Co-Occurrence). Let $\mathcal{D}$ be a collection of process graphs with disjoint sets of nodes, $\mathcal{N}$ be the node set of $\mathcal{D}$, and $\omega(l)$ is the function that maps a label $l$ to the set of words that appear in $l$.

- *Frequency of Word Co-Occurrence (FWCO)*: Given a word $w$ and another word $w_1$ ($w_1 \neq w$), the frequency of the word co-occurrence of $w$ and $w_1$ is the number of nodes of the collection $\mathcal{D}$ that contain both $w$ and $w_1$. Formally, $\text{FWCO}(w, w_1) = \vert \{n \in \mathcal{N} | w, w_1 \in \omega(\lambda(n)) \} \vert$. We say that word $w_1$ co-occurs in the same node label with $w$ if $\text{FWCO}(w, w_1) > 0$.

- *Frequency of Pre-Word Co-Occurrence (FWCO_pre)*: Given two words $w$ and $w_1$, the frequency of the pre-word co-occurrence of $w_1$ with respect to $w$ is the number of process fragments (a sequence of two nodes) satisfying that $w$ appears in the label of a node $n$, $w_1$ appears in the label of a node $n_1$, and there exists a process graph $g$, in which $n_1$ is in the pre-set of $n$. Formally, $\text{FWCO_pre}(w, w_1) = \vert \{n_1 \in \mathcal{N} | g \in \mathcal{D} \land n, n_1 \in N_g \land n_1 \in \bullet n \land w \in \omega(\lambda(n)) \land w_1 \in \omega(\lambda(n_1)) \} \vert$. We say that word $w_1$ co-occurs with $w$ in a pre-set node label if $\text{FWCO_pre}(w, w_1) > 0$.

- *Frequency of Post-Word Co-Occurrence (FWCO_post)*: Given two word $w$ and $w_1$, the frequency of the post-word co-occurrence of $w_1$ with respect to $w$ is the number of process fragments (a sequence of two nodes) satisfying that $w$ appears in the label of a node $n$, $w_1$ appears in the label of a node $n_1$, and there exists a process graph $g$, in which $n_1$ is in the post-set of $n$. Formally, $\text{FWCO_post}(w, w_1) = \vert \{n_1 \in \mathcal{N} | g \in \mathcal{D} \land n, n_1 \in N_g \land n_1 \in n \bullet w \land w \in \omega(\lambda(n)) \land w_1 \in \omega(\lambda(n_1)) \} \vert$. We say that word $w_1$ co-occurs with $w$ in a post-set node label if $\text{FWCO_post}(w, w_1) > 0$.

For example, with respect to the word ‘receive’ in the collection in Figure 8.1, the frequency of the word co-occurrence for ‘goods’ is 4; the frequency of pre-word co-occurrence for ‘goods’ is 4; the frequency of post-word co-occurrence for ‘goods’ is 1.

The probabilities of the three types of word co-occurrences are defined in Definition 8.4.

**Definition 8.4** (Probabilities of Word Co-Occurrences). Let $\mathcal{D}$ be a collection of process graphs with disjoint sets of nodes, $\mathcal{N}$ be the node set of $\mathcal{D}$, $\mathcal{W}$ be the word
set of $D$, and $\omega(l)$ is the function that maps a label $l$ to the set of words that appear in $l$. Three types of word co-occurrence probabilities are defined as follows.

- **Probability of Word Co-Occurrences (PWCO):** Given a word $w$ and another word $w_1$, the probability of word co-occurrences of $w$ and $w_1$ is the frequency of $w_1$ co-occurring with $w$ divided by the frequency of all words co-occurring with $w$. Formally,

  \[
  PWCO(w, w_1) = \frac{FWCO(w, w_1)}{\sum_{w_2 \in W \land w_2 \neq w} FWCO(w, w_2)}.
  \]  

- **Probability of Pre Word Co-Occurrence (PWCO\_pre):** Given two words $w$ and $w_1$, the probability of the co-occurrences of $w_1$ with respect to $w$ is the frequency of $w_1$ co-occurring with $w$ in a pre-set node label divided by the frequency of all words co-occurring with $w$ in a pre-set node label. Formally,

  \[
  PWCO\_pre(w, w_1) = \frac{FWCO\_pre(w, w_1)}{\sum_{w_2 \in W} FWCO\_pre(w, w_2)}.
  \]

- **Probability of Post Word Co-Occurrence (PWCO\_post):** Given two words $w$ and $w_1$, the probability of the co-occurrences of $w_1$ with respect to $w$ is the frequency of $w_1$ co-occurring with $w$ in a post-set node label divided by the frequency of all words co-occurring with $w$ in a post-set node label. Formally,

  \[
  PWCO\_post(w, w_1) = \frac{FWCO\_post(w, w_1)}{\sum_{w_2 \in W} FWCO\_post(w, w_2)}.
  \]

For example, with respect to the word ‘receive’ in the collection in Figure 8.1, the probability of word co-occurrence for ‘goods’ is $4/5=0.80$; the probability of pre-word co-occurrence for ‘goods’ is $4/9=0.44$; the probability of post-word co-occurrence for ‘goods’ is $1/3=0.33$.

**Structural Properties**

The structure of a process model can be described in terms of a set of common patterns, which are called features in this thesis. In this section, properties of these features are considered as structural properties, e.g., feature type and feature size as defined in Definition 5.2. Furthermore, the composition rules are abstracted, which indicate how features can be composed to form a process graph.

As defined in Definition 5.2, four types of structural features are considered for the purpose of process retrieval, i.e., sequence, split, join, and loop. Process graphs
8.2. Synthetic Process Model Generator

consists of compositions of these features. For example, graph 6 of Figure 8.2 consists of a sequence feature \( a \rightarrow b \rightarrow c \), a split feature \( b \rightarrow \{c, d, e, f, g\} \); a join pattern \( \{c, d\} \rightarrow h \), a loop feature \( f \rightarrow k \), etc.

Figure 8.2: A Business Process Graph

When splitting a process graph into features, a feature can be a parent of different features (parent and child features are defined in Definition 5.3). For example, in graph 6 of Figure 8.2, sequence feature \( b \rightarrow c \) is a parent of sequence features \( a \rightarrow b \rightarrow c \rightarrow h \), \( b \rightarrow c \rightarrow h \), etc. This affects the probability of the occurrence of a certain type of feature. For example, sequence features are counted too many times, since all structural features consists of sequence features of size 1. Therefore, only local maximal features are considered, which are features without child features, as defined in Definition 8.5. For example, sequence feature \( b \rightarrow c \) is not considered any more; while sequence feature \( a \rightarrow b \rightarrow c \rightarrow h \) is considered.

**Definition 8.5 (Local Maximal Feature).** Let \( g = (N, E, \lambda) \) be a business process graph. Let \( f \), a subgraph of \( g \), be a feature. The feature is a local maximal feature of \( g \), denoted as \( f \in LMF(g) \), if and only if there is not another feature \( f_1 \) of \( g \), such that \( f \) is a parent feature of \( f_1 \).

In a process graph, a node can be in different local maximal features and have different pre-sets or post-sets in these features. These nodes are called open nodes. In Section 8.2.2, when generating process graphs, we use these nodes as the points to extend a process graph and create a large synthetic graph. An open node is defined as follows.
**Definition 8.6 (Open Node, Closed Node).** Let $g$ be a process graph and $f$ be a feature of $g$. A node $n$ of the feature $f$ is an open node for that feature, if and only if the feature does not contain all of the nodes in the pre-set or post-set of $n$. Formally, $\forall n \in N_f : n \in ON(f) \iff \exists((\bullet n \cup n \bullet) - N_f) \neq \emptyset$. A node in the feature is a pre-open (post-open) node for the feature, if and only if the feature does not contain all of the nodes in the pre-set (post-set) of $n$, denoted as $n \in ON_{pre}(f) (n \in ON_{post}(f))$. A node is a closed node if it is not an open node.

For example, in graph 6 of Figure 8.2, node $d$ is a post-open node for split feature $b \rightarrow \{c,d,e,f,g\}$, and node $d$ is both a pre-open and post-open node for join feature $\{c,d\} \rightarrow h$. When generating a synthetic graph, given $b \rightarrow \{c,d,e,f,g\}$ is already in the synthetic graph, node $d$ can be used to extend the graph with another feature that has a pre-open node. More details about extending a synthetic graph is given in Section 8.2.2.

For now, we know which node in a feature is an open node that can be used to extend another feature. However, for a split or join feature, a subset of open nodes can be associated to extend features. The split or join association of open nodes is to support a structure like that after a split some of the branches join together immediately or eventually. For example, in Figure 8.2 after the split node $b$, the branches of node $c$ and node $d$ join together immediately at node $h$; the branches of node $e$ and $f$ join together eventually at node $j$. The split or join association of open nodes is defined in Definition 8.7.

**Definition 8.7 (Split (Join) Association of Open Nodes).** Let $g$ be a business process graph and $f$ be a split feature. Let the node $n \in N_f$ be the split node and let $N' = ON_{post}(f)$ be the post-open node set for $f$. Let $N'' \subseteq N'$ ($|N''| > 1$) be a subset of the post-open nodes. If there exists a node $n_2 \in N_g$, such that for each $n_1 \in N''$, $(n_1,n_2) \in E_g$, we say that $f$ has an immediate join, denoted as $\zeta(N'') = Im$; otherwise if there exists a node $n_2 \in N_g$, such that for each $n_1 \in N''$, there exists a sequence feature in $g$, $n_1 \rightarrow \ldots \rightarrow n_2$, we say that $f$ has an eventual join, denoted as $\zeta(N'') = Ev$. $Im$ and $Ev$ are the type of a split (join) association of open nodes. The join association of open nodes is defined similarly.

For example, Figure 8.3 shows some split and join features of graph 6 in Figure 8.2, the split feature $b \rightarrow \{c,d,e,f,g\}$ has an immediate join $\{c,d\}$ and an eventual join $\{d,e\}$; while the join feature $\{c,d\} \rightarrow h$ has an immediate split $\{c,d\}$ and the join feature $\{d,i\} \rightarrow j$ has an eventual split $\{d,i\}$. For an immediate join
(split), a join (split) feature is inserted to extend the process graph; for an event-
tual join (split), a join (split) feature and several sequence features are inserted to
extend the process graph.

![Diagram](image)

Legend: ○ represents a open node; □ represents that the
open nodes in it are associated.

**Figure 8.3:** Examples of Join and Split Associations of Open Nodes of *graph 6* in Figure 8.2

We know that a feature can be inserted into a process graph if they both have
open nodes or split (join) associations of open nodes. However, the definitions of a
feature and a process graph do not contain items to indicate which nodes are open
nodes or split (join) associations of open nodes. Therefore, we define a component
based on a feature of a process graph to record this information. We need this
information, because in next section components are composed to form a (partial)
synthetic process graphs, and open nodes or split (join) associations of open nodes
indicate how to compose components. A component is defined as follows.

**Definition 8.8** (Component). Let $g$ be a business process graph and let $f$ be a local
maximal feature of $g$. The component of $f$, denoted as $c = \text{Comp}(f, g)$, is a tuple
$(N, E, \lambda, \text{preON}, \text{postON}, \zeta)$, in which:

- $N = N_f$ is the set of nodes.
- $E \subseteq N \times N$ is the set of edges, where $E = E_f$.
- $\lambda$ is a function that maps each node in $N$ to an empty label.
- $\text{preON} = \{n \in N_f | (\bullet n - N_f) \neq \emptyset\}$ is the pre-open node set.
- $\text{postON} = \{n \in N_f | (n \bullet - N_f) \neq \emptyset\}$ is the pre-open node set.
• $\zeta$ is a function that maps a subset of the pre-open (post-open) node set to an immediate or eventual join (split), as defined in Definition 8.7.

The size and type of a component are the same as the size and type of the feature it derives from, i.e., $\text{Size}(s) = \text{Size}(f)$ and $\text{Type}(s) = \text{Type}(f)$.

For example, $\{(a, b, g), ((a, b), (b, g)), \lambda, \emptyset, \{b\}, \zeta\}$ is a sequence component of graph 6 (labels are used to identify nodes here). Definition 8.9 presents how to abstract all components from a process graph or a collection of process graphs.

Definition 8.9 (Component Set). Let $D$ be a collection of process graphs. The component set of $D$ consists of the components of all local maximal of process graphs in $D$. Formally, $C = \{\text{Comp}(f, g)|g \in D \land f \in \Gamma(g) \cap \text{LMT}(g)\}$.

We say two components are equivalent if there is a mapping between two components, as defined in Definition 8.10.

Definition 8.10 (Component Equivalence). Let $c_1 = (N_1, E_1, \lambda, \text{preON}_1, \text{postON}_1, \zeta)$ and $c_2 = (N_2, E_2, \lambda, \text{preON}_2, \text{postON}_2, \zeta)$ be two components. Components $c_1$ and $c_2$ are equivalent, denoted as $c_1 = c_2$, if and only if there exists a one-to-one mapping $M : N_1 \to N_2$, such that

- $\forall n \in \text{preON}_1: M(n) \in \text{preON}_2; \forall n \in \text{postON}_1: M(n) \in \text{postON}_2; \forall n \in (N_1 - \text{preON}_1 - \text{postON}_1): M(n) \in (N_2 - \text{preON}_2 - \text{postON}_2)$;
- $\forall (n, m) \in E_1: (M(n), M(m)) \in E_2$;
- if $s_1$ and $s_2$ are split (join) components, $\forall sN_1 \in \text{postON}_1(\text{preON}_1): \zeta_1(sN_1) = \zeta_2(\{M(n_1)|n_1 \in sN_1\})$.

For example, in Figure 8.3 components $\{c, d\} \to h$ and $\{d, i\} \to j$ are not equivalent, because $\zeta(\{c, d\}) \neq \zeta(\{d, i\})$. Based on the component equivalence, the frequency and probability of component occurrence are defined as follows.

Definition 8.11 (Frequency of Component Occurrence, Probability of Component Occurrence). Let $D$ be a collection of process graphs and $C$ be the component set of $D$. Let $c \in C$ be a component. The frequency of the occurrence of $c$ is the number of local maximal features in $D$ having the equivalent component with $c$, denoted as $\text{FCO}(c, C) = |\{f \in \Gamma(g)|g \in D \land f \in \text{LMF}(g) \land \text{Comp}(f, g) = c\}|$; the probability of its occurrence is the fraction between the frequency of its occurrence and the frequency of the occurrence of all components, denoted as $\text{PCO}(c, C) = \frac{\text{FCO}(c, C)}{\sum_{c_1 \in C} \text{FCO}(c_1, C)}$. 
8.2. Synthetic Process Model Generator

8.2.2 Generating Synthetic Process Model Collections

This section presents the algorithm to generate synthetic process models based on the properties defined in the previous section. The algorithm consists of two steps. It first generates node labels and it then generates a synthetic graph by inserting components into the graph and labeling component nodes.

Synthetic labels can be generated based on the probabilities of word occurrence and word co-occurrence, which is defined in Definition 8.12.

**Definition 8.12 (Synthetic Label).** Let $\mathcal{D}$ be a collection of process graphs with disjoint sets of nodes, $\mathcal{N}$ be the node set of $\mathcal{D}$, $\mathcal{W}$ be the word set of $\mathcal{D}$, and $\omega(l)$ be the function that maps a label $l$ to the set of words that appear in $l$.

A synthetic label of size $s$ consists of a word, $w \in \mathcal{W}$, and a set of $s - 1$ words, $\mathcal{W} = \{w_1, w_2, ..., w_{s-1}\} \subset \mathcal{W}$ ($w \notin \mathcal{W}$), which co-occur with $w$.

The probability of the size $s$ is $\frac{|\{n \in \mathcal{N} | \omega(\lambda(n)) = s\}|}{|\mathcal{N}|}$. The probability of selecting a word $w$ is $PWO(w)$, according to Definition 8.2. The probability of selecting a word $w_i \in \mathcal{W}$ is $PWCO(w, w_i)$, according to Definition 8.3.

A synthetic node is labeled by selecting a synthetic label from a set of synthetic labels. The selection consists of two steps. Firstly, a word is selected based on the probabilities of pre-word and post-word co-occurrence. Second, a synthetic label is selected from the subset of synthetic labels containing the selected word. The probability of selecting a label is defined as follows.

**Definition 8.13 (Probability of Label Selection).** Let $\mathcal{SL}$ a set of synthetic labels generated according to Definition 8.12, let $sl$ be a synthetic, $sl \in \mathcal{SL}$, and let $\mathcal{W} = \{w | w \in \omega(sl) \land sl \in \mathcal{SL}\}$ be the word set of $\mathcal{SL}$. Let $c$ be a component and let $n$ be a node of $c$, $n \in \mathcal{N}_c$.

The probability of selecting a word $w$ is normally the frequency of words in the pre-set/post-set of $n$ co-occurs with $w$ in a pre-set/post-set node label divided by the frequency of words in the pre-set/post-set of $n$ co-occurs with any word in a pre-set/post-set node label; however, if the denominator is 0, the probability of selecting $w$ is the probability of the occurrence of $w$. 
Chapter 8. Evaluation

\[
PLS(w) = \begin{cases} 
\frac{\sum_{w' \in W_{post}} FWCO_{pre}(w', w) + \sum_{w' \in W_{pre}} FWCO_{post}(w', w)}{\sum_{w' \in W(W)} \frac{FWCO_{pre}(w', w) + \sum_{w' \in W_{pre}} FWCO_{post}(w', w')}}; \\
PWO(w), \; \text{otherwise.}
\end{cases}
\]

where \( W_{post} = \{ w | \forall n_1 \in n \cdot \land w \in \omega(\lambda(n_1)) \} \) and \( W_{pre} = \{ w | \forall n_1 \in n \cdot \land w \in \omega(\lambda(n_1)) \} \).

The probability of selecting a synthetic label \( sl \) from \( SL \) based on \( w \) is one divided by the number of labels that contains the word \( w \), i.e., \( P_l(w, sl) = \frac{1}{|\{ sl_1 \in SL | w \in \omega(sl_1) \}|} \).

Overall, the probability of selecting a label \( sl \) from \( SL \) is \( P(sl) = \sum_{w \in \omega(sl)} (P_w(w) \times P_l(w, sl)) \).

To generate a synthetic graph, the size of the synthetic graph is required, of which the probability is defined as follows.

**Definition 8.14** (Probability of Graph Size). Let \( D \) be a collection of process graphs. The probability of a graph \( g \) of size \( s \) in \( D \), \( PGS(s, D) \), is that the number of graphs of size \( s \) in \( D \) divided by the number of all graphs in \( D \). Formally, \( PGS(s, D) = \frac{|\{ g \in D | |E_g| = s \}|}{|D|} \).

Algorithm 8 presents the algorithm for generating synthetic process graphs. Firstly, a set of synthetic labels are generated, according to Definition 8.12, which are used to label nodes in components later (line 2). Then, a set of synthetic process graphs are generated. Each graph is generated as follows. Initially a synthetic process graph \( sg \) contains one node \( n \) that is both a pre-open and post-open node \( (n \in preON \cap postON) \); the label of \( n \) is empty; The synthetic process graph does not contain any edge (line 6). Then components are inserted to extend the graph (lines 8-21).

To insert a component into a synthetic graph, an open node is randomly selected. If the open node is not in join (split) associations, a component is selected, having an open node that can be merged with the open node in the synthetic graph as explained in Algorithm 9. If the open node is in join (split) associations, one of the join (split) associations is randomly selected. Then, a component is selected, which has a join (split) association with the same number of open nodes and the same type (line 19); the probability of the selection is the probability is \( PCO(c, C_1) \) as
8.3  Evaluation for Querying

This section shows how the use of the technique in this thesis affects process querying in terms of performance and quality. The experiments were run with both real-life and synthetic process models. We use Definition 6.1 to match nodes for process querying in the experiments.

explained in Definition [8.11]. Finally, the open nodes in the join (split) associations are merged as explained in Algorithm [10].

Algorithm [9] presents the steps of merging an open node of a synthetic graph and an open node of a component. Firstly, (open) nodes in the component are inserted the (open) node set of the synthetic graph; the open nodes to be merged are not open nodes anymore. Secondly, edges connected to the open node in the component connect to the open node in the synthetic graph; other edges in the component are inserted into the edge set of the synthetic graph. Thirdly, nodes in the component expect for the open node are labeled according to Definition [8.13].

Algorithm [10] presents the steps of merging a join (split) association of open nodes of a synthetic graph and a split (join) association of open nodes of a component. If the associations are immediate, the pair of nodes are merged (Algorithm [9]). If the associations are eventual, for each pair open nodes (one in the synthetic graph and one in the component) a sequence component is selected to connect the pair of nodes. The connection is done by merging the pair of open nodes with two open nodes in the sequence component respectively (lines [7][11]). The join (split) associations of open nodes are updated. The merged associations are deleted. If there are other split and join associations in the components, these associations are recorded in the synthetic graph (lines [12][15]).

Taking process graphs transformed from the 604 SAP reference process models as input, we generated a collection of 6040 synthetic process graphs with the algorithm. On average, a SAP reference process graph contains 20.7 nodes and 20.5 edges; on average, a synthetic process graph contains 20.3 nodes and 24.1 edges. We can see that the average numbers of nodes (edges) in these two collections are close. The synthetic graph has more edges, which is expected, because when we only stop composing a synthetic graph with components when its size is not less than a selected size of a SAP reference process graph.
Algorithm 8: Synthetic Process Graph Generation

**input**: a collection of process graphs: \( \mathcal{D} \), an integer: \( size_c \)

**output**: a collection of synthetic process graphs \( SD \)

1. \( S\mathcal{L} \) is the set of generated synthetic labels (Definition 8.12);
2. \( SD \leftarrow \emptyset \);
3. \( C \) is the component set of \( \mathcal{D} \) (Definition 8.9);
4. while \( |SD| < size_c \) do
5. \hspace{1em} select \( sg = \{\{sn\},\emptyset,\emptyset,\{sn\},\emptyset\} \); //sn is a newly created node.
6. \hspace{1em} select \( size_g \in \{|E_g|, g \in \mathcal{D}\} \) with probability \( PGS(size_g, \mathcal{D}) \) (Definition 8.14);
7. \hspace{1em} while \( |E_{sg}| < size_g \land (preON_{sg} \cup postON_{sg}) \neq \emptyset \) do
8. \hspace{2em} randomly select \( n \in preON_{sg} \cup postON_{sg} \);
9. \hspace{2em} if \( n \in postON_{sg} \) then
10. \hspace{3em} if \( \forall N \subseteq N_{sg} : n \in N \land \zeta_{sg}(N) \in \{Im, Ev\} \) then
11. \hspace{4em} \( C_1 \leftarrow \{c \in C|preON_c \neq \emptyset\} \);
12. \hspace{4em} select \( c \in C_1 \) with probability \( PCO(c, C_1) \) (Definition 8.11);
13. \hspace{4em} randomly select \( n_1 \in preON_c \);
14. \hspace{4em} \( sg \leftarrow \text{mergeNode}(sg, c, n, n_1, S\mathcal{L}) \); //Algorithm 9
15. \hspace{2em} else
16. \hspace{3em} randomly select
17. \hspace{4em} \( AN \in \{N \subseteq N_{sg}|n \in N \land \zeta_{sg}(N) \in \{Im, Ev\}\} \);
18. \hspace{4em} \( C_1 \leftarrow \{c \in C|Type(c) = join \land AN \subseteq preON_c \land |AN| = |AN_1| \land \zeta_{sg}(AN) = \zeta_c(AN_1)\} \);
19. \hspace{4em} select \( c \in C_1 \) with probability \( PCO(c, C_1) \) (Definition 8.11);
20. \hspace{4em} \( sg \leftarrow \text{mergeAsso}(sg, c, AN, AN_1, S\mathcal{L}) \); //Algorithm 10
21. \hspace{2em} if \( n \in preON_{sg} \) then //similar to lines 11-20
22. \hspace{3em} \( SD \leftarrow SD \cup \{sg\} \);
23. \hspace{1em} return \( SD \);
Algorithm 9: Merge a Pair of Open Nodes

**input**: a synthetic process graph: \( sg \), a component: \( c \), an open node of \( sg \): \( n_{sg} \), an open node of \( c \): \( n_{c} \), a synthetic label set: \( SL \)

**output**: a synthetic process graphs: \( sg \)

1. \( N_{sg} \leftarrow N_{sg} \cup (N_{c} - \{n_{c}\}) \);
2. if \( n_{sg} \in preON_{sg} \land n_{c} \in postON_{c} \) then
   3. \( preON_{sg} \leftarrow (preON_{sg} - \{n_{sg}\}) \cup preON_{c} \);
   4. \( postON_{sg} \leftarrow postON_{sg} \cup (postON_{c} - \{n_{c}\}) \);
5. else if \( n_{sg} \in postON_{sg} \land n_{c} \in preON_{c} \) then
   6. \( preON_{sg} \leftarrow preON_{sg} \cup (preON_{c} - \{n_{c}\}) \);
   7. \( postON_{sg} \leftarrow (postON_{sg} - \{n_{sg}\}) \cup postON_{c} \);
8. \( E_{sg} \leftarrow E_{sg} \cup \{(n, n_{3}) | n_{3} \in n_{1} \bullet \} \cup \{(n_{3}, n) | n_{3} \in \bullet n_{1} \} \)
9. \( \cup (E_{c} - \{(n_{1}, n_{3}) | n_{3} \in n_{1} \bullet \} - \{(n_{3}, n_{1}) | n_{3} \in \bullet n_{1} \} \); 
10. foreach \( n \in (postON_{c} - \{n_{c}\}) \) do
    11. select a label \( sl \) from \( SL \) with probability \( P(sl) \) (Definition 8.13).
    12. \( \lambda_{sg} \leftarrow \lambda_{sg} \cup \{(n, sl)\} \);
13. return \( sg \);

8.3.1 Real-life Process Models

In this subsection, we present the experiments with real-life process models. We first explain the setup of the evaluation and then the results.

**Evaluation Setup**

The experiments with real-life process models were performed on the collection of SAP reference models. Two groups of queries were designed. The first group consists of queries of smaller size and the second group consists of queries of bigger size, as such we can see whether the size of query models influences the efficiency of process querying. The properties of queries are summarized in Table 8.1. In the first group, as shown in Figure 8.4, five queries were adapted from the evaluation of BPMN-Q [6]. Query a, b, d, e, and g in the evaluation of [6] were selected. Query c and f were not selected, because they test specific types of elements in BPMN-
Algorithm 10: Merge Join and Split Associations of Open Nodes

**input**: a synthetic process graphs: $sg$, a component: $c$, a join (split) association of $sg$: $AN_{sg}$, a split (join) association of $c$: $AN_c$, a synthetic label set: $SL$

**output**: a synthetic process graphs: $sg$

```
begin
	foreach $n_{sg} \in AN_{sg}$ do
		randomly select $n_c$ from $AN_c$;
		$AN_c \leftarrow AN_c - \{n_c\}$;
		if $\zeta_{sg}(AN_{sg}) = \text{Im}$ then
			$sg \leftarrow \text{mergeNode} (sg,c,n_{sg},n_c,SL)$;//Algorithm 9
	else if $\zeta_{sg}(AN_{sg}) = \text{Ev}$ then
			$C_1 \leftarrow \{c_1 \in C | \text{Type}(c_1) = \text{sequence} \land (\exists (n_1 \in \text{preON}_{c_1} \land n_2 \in \text{postON}_{c_1}), n_1 \rightarrow \ldots \rightarrow n_2 \text{ is a subgraph of } c_1)\};$
			select $c_1 \in C_1$ with probability $FCO(c_1,C_1)$ (Definition 8.11);
			$sg \leftarrow \text{mergeNode} (sg,c_1,n_{sg},n_1,SL)$;//Algorithm 9
			$sg \leftarrow \text{mergeNode} (sg,c,n_2,n_c,SL)$;//Algorithm 9
	Delete $AN_{sg}$ from the domain of $\zeta_{sg}$;
	Delete $AN_c$ from the domain of $\zeta_c$;
	foreach $AN_1 \in \{N_1 \subseteq N_c | \zeta_c(N_1) \in \{\text{Im}, \text{Ev}\}\}$ do
		$\zeta_{sg}(AN_1) \leftarrow \zeta_c(AN_1)$;
return $sg$;
```

Q that are simply considered nodes in this thesis and are not handled differently from other types of nodes. Each node of the queries was labeled by one or two words from the original labels of these SAP reference models, such that the queries have matches in the collection of SAP reference models. More precisely, *query 1* is a sequence of size 2, composed of three basic nodes and two basic edges; *query 2* is a sequence of size 1, composed of a basic node, a wildcard node and a basic edge; *query 3* is a join of size 2, composed of three basic nodes and two transitive edges; *query 4* added a neg-transitive edge in *query 3* between the two nodes before joining; *query 5* is a loop of size 1, composed of a node and a transitive edge.
8.3. Evaluation for Querying

The second group were larger query models. The models in this group were designed by randomly selecting 5 from the collection of SAP reference models; then the models were adapted such that each query in this group contains as many advanced constructs as the corresponding query in the first group. *query 6* is composed of 9 basic nodes and 8 basic edges; *query 7* contains 1 wildcard node besides 22 basic nodes and 22 basic edges; *query 8* contains 4 transitive edges besides 22 basic nodes and 17 basic edges; *query 9* contains 2 neg-transitive edges besides 27 basic nodes and 26 basic edges; besides 20 basic nodes and 23 basic edges, *query 10* contains 1 transitive edge with the same source and target node (a loop of size 1).

<table>
<thead>
<tr>
<th>Query Name</th>
<th>Basic Elements</th>
<th>Advanced Elements</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Group 1</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query 1</td>
<td>3 nodes, 2 edges</td>
<td>None</td>
<td>7</td>
</tr>
<tr>
<td>Query 2</td>
<td>1 node, 1 edge</td>
<td>1 wildcard node</td>
<td>13</td>
</tr>
<tr>
<td>Query 3</td>
<td>3 nodes</td>
<td>2 transitive edges</td>
<td>10</td>
</tr>
<tr>
<td>Query 4</td>
<td>3 nodes, 1 edge</td>
<td>1 transitive, 1 neg-transitive edges</td>
<td>10</td>
</tr>
<tr>
<td>Query 5</td>
<td>1 node</td>
<td>1 transitive edge (loop)</td>
<td>10</td>
</tr>
<tr>
<td><strong>Group 2</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query 6</td>
<td>9 nodes, 8 edges</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>Query 7</td>
<td>22 nodes, 22 edges</td>
<td>1 wildcard node</td>
<td>3</td>
</tr>
<tr>
<td>Query 8</td>
<td>22 nodes, 17 edges</td>
<td>4 transitive edges</td>
<td>1</td>
</tr>
<tr>
<td>Query 9</td>
<td>27 nodes, 26 edges</td>
<td>2 neg-transitive edges</td>
<td>2</td>
</tr>
<tr>
<td>Query 10</td>
<td>20 nodes, 23 edges</td>
<td>1 transitive edge (loop)</td>
<td>1</td>
</tr>
</tbody>
</table>

*Figure 8.4: First Group of Queries*
Chapter 8. Evaluation

Evaluation Results

To evaluate the performance of the technique, both groups of queries were used to run experiments. Table 8.2 shows the results of performing the queries on the collection of SAP reference models. The columns in the table show the execution time of each query and the average total time over the 5 queries in each group. The rows in the table show the features that are used to construct a PFIndex. In the first row process querying is performed based on node features of size 0 (N(0)), sequence features of size 1 (S(1)), and loop features of size 1 (L(1)). In the second row process querying is performed based on features in the first row (1) and sequence features of size 2 (S(2)) and loop features of size 2 (L(2)). Features of the rows of Group 2 are described similarly.

In Table 8.2 we can see that on average a query in Group 1 is performed in 0.03 second and a query in Group 2 is performed in 0.06 second. The execution time of the first group is faster than the second group on average. This is because in the second group, there are more basic nodes and edges in the query models and more words in the node labels, therefore more feature comparisons are required. The second group has fewer matches than the first group. On average, for each query in the first group there are 10 hits from the collection of SAP reference models; while on average there are 1.6 positive results for each query in the second group (Table 8.1).

To better evaluate the performance of our technique, we compared it with the performance of BPMN-Q [10]. It on average takes 5s to perform a query with a collection of 500 process models (each model on average has 12 nodes) on a PC (2.8 GHz processors and 4GB memory); while a PFIndex on average takes 0.045s to perform a query with a collection of 604 process models (each model on average has 20.7 nodes) on a laptop (2.2 GHz processors and 4GB memory). The queries in this thesis have similar characteristics as in [10] in terms of the number and type of advanced query elements. From the comparison we conclude that on average the technique performs two orders of magnitude faster than BPMN-Q.

To evaluate the quality of the technique, the first group of queries were used to run an experiment. The collection was developed in two steps. First, twenty SAP reference models were selected. We manually checked each pair of query and selected SAP reference model to see whether the selected SAP reference model is a positive or negative result for the query. We make sure that each query at least
Table 8.2: Execution Time

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Features(size)</th>
<th>query1</th>
<th>query2</th>
<th>query3</th>
<th>query4</th>
<th>query5</th>
<th>T_{avg}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1:N(0)+S(1)+L(1)</td>
<td>0.04s</td>
<td>0.07s</td>
<td>0.01s</td>
<td>0.01s</td>
<td>0.02s</td>
<td>0.03s</td>
</tr>
<tr>
<td></td>
<td>2:1+S(2)+L(2)</td>
<td>0.04s</td>
<td>0.08s</td>
<td>0.01s</td>
<td>0.01s</td>
<td>0.02s</td>
<td>0.03s</td>
</tr>
<tr>
<td></td>
<td>3:1+Split(2)</td>
<td>0.04s</td>
<td>0.07s</td>
<td>0.004s</td>
<td>0.01s</td>
<td>0.02s</td>
<td>0.03s</td>
</tr>
<tr>
<td></td>
<td>4:3+Split(3)</td>
<td>0.04s</td>
<td>0.09s</td>
<td>0.01s</td>
<td>0.01s</td>
<td>0.03s</td>
<td>0.03s</td>
</tr>
<tr>
<td></td>
<td>5:1+Join(2)</td>
<td>0.04s</td>
<td>0.07s</td>
<td>0.01s</td>
<td>0.01s</td>
<td>0.02s</td>
<td>0.03s</td>
</tr>
<tr>
<td></td>
<td>6:5+Join(3)</td>
<td>0.04s</td>
<td>0.07s</td>
<td>0.01s</td>
<td>0.01s</td>
<td>0.02s</td>
<td>0.03s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group 2</th>
<th>Features(size)</th>
<th>query6</th>
<th>query7</th>
<th>query8</th>
<th>query9</th>
<th>query10</th>
<th>T_{avg}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1:N(0)+S(1)+L(1)</td>
<td>0.002s</td>
<td>0.09s</td>
<td>0.03s</td>
<td>0.06s</td>
<td>0.12s</td>
<td>0.06s</td>
</tr>
<tr>
<td></td>
<td>2:1+S(2)+L(2)</td>
<td>0.002s</td>
<td>0.10s</td>
<td>0.04s</td>
<td>0.06s</td>
<td>0.13s</td>
<td>0.07s</td>
</tr>
<tr>
<td></td>
<td>3:1+Split(2)</td>
<td>0.002s</td>
<td>0.10s</td>
<td>0.03s</td>
<td>0.06s</td>
<td>0.13s</td>
<td>0.06s</td>
</tr>
<tr>
<td></td>
<td>4:3+Split(3)</td>
<td>0.004s</td>
<td>0.19s</td>
<td>0.03s</td>
<td>0.06s</td>
<td>0.92s</td>
<td>0.24s</td>
</tr>
<tr>
<td></td>
<td>5:1+Join(2)</td>
<td>0.002s</td>
<td>0.10s</td>
<td>0.03s</td>
<td>0.06s</td>
<td>0.13s</td>
<td>0.06s</td>
</tr>
<tr>
<td></td>
<td>6:5+Join(3)</td>
<td>0.001s</td>
<td>0.09s</td>
<td>0.03s</td>
<td>0.06s</td>
<td>0.13s</td>
<td>0.06s</td>
</tr>
</tbody>
</table>

N=Node, S=Sequence, and L=Loop.

have one positive result within the two models. Second, for each query, three models were artificially made to check if the technique in this thesis works well in terms of result quality. One of the model is a positive result for the query and the other two are negative results for the query. The experiment results show that both precision and recall of the technique are 1.

8.3.2 Synthetic Process Models

In this subsection, we present the experiments with synthetic process models. We first explain the setup of the evaluation and then the results.

Evaluation Setup

To investigate the execution time of the technique when the size of collections increases, we generated 6040 synthetic process models using the generator described in Section 8.2. We performed two experiments, running five queries with a collection
604 synthetic models and a collection of 6040 synthetic models. These five query models were designed based on the first group of query models in Section 8.4.1. For each query models in the first group of query models in Section 8.4.1 a synthetic model was randomly selected from the 604 synthetic models; then the label of the query model was replaced by label words in the selected synthetic models, such that the query is matching with the selected model after replacing the labels to make sure there are matching models for the query model in the collection. The results of the former experiment are used to compare with the results of the experiment with 604 SAP reference models, which evaluates whether the process querying technique in this thesis works well with different collections. The results of the latter experiment are used to compare with the results of the former one, which evaluates the scalability of the process querying technique in this thesis.

**Evaluation Results**

Table 8.3 shows the execution time of the process querying running with 604 synthetic models. The structure of the table is the same as Table 8.2. The average execution time of a query is 0.02s; while the average execution time of a query running with 604 SAP models is 0.03s. By comparison, we can see that the execution times are close. Although querying with synthetic models takes less time than querying with SAP model. This should be because the different characteristics of models in these two collections. For example, the collection of synthetic models contain 12280 nodes and 2598 different labels; while the collection of SAP models contain 12529 nodes and 3795 different labels. As such, the PFIndex of the synthetic models contains less node features and each node feature maps to more nodes with identical labels, which makes the indexing power more evident.

Table 8.4 shows the execution time of the process querying running with 6040 synthetic models. The average execution time of a query is 0.10s. By comparing the execution time of 604 synthetic models, we can see that in the latter experiment the technique performs five times slower than the former experiment when the size of the collection is ten times bigger. This is expected, because when the collection size increases, the efficiency of querying through the technique with an index (PFIndex) should be better than a linear algorithm. Using BPMN-Q [10], it on average takes about 6s to execute a query with 1000 models. We conclude that the technique works well when the size of the collection increases.
8.4. Evaluation for Process Similarity Search

### Table 8.3: Execution Time Using 604 Synthetic Process Models

<table>
<thead>
<tr>
<th>Features(size)</th>
<th>query1</th>
<th>query2</th>
<th>query3</th>
<th>query4</th>
<th>query5</th>
<th>$T_{avg}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:N(0)+S(1)+L(1)</td>
<td>0.002s</td>
<td>0.05s</td>
<td>0.01s</td>
<td>0.02s</td>
<td>0.03s</td>
<td>0.02s</td>
</tr>
<tr>
<td>2:1+S(2)+L(2)</td>
<td>0.001s</td>
<td>0.05s</td>
<td>0.01s</td>
<td>0.01s</td>
<td>0.02s</td>
<td>0.02s</td>
</tr>
<tr>
<td>3:1+Split(2)</td>
<td>0.001s</td>
<td>0.05s</td>
<td>0.003s</td>
<td>0.01s</td>
<td>0.02s</td>
<td>0.02s</td>
</tr>
<tr>
<td>4:3+Split(3)</td>
<td>0.001s</td>
<td>0.06s</td>
<td>0.01s</td>
<td>0.02s</td>
<td>0.02s</td>
<td>0.02s</td>
</tr>
<tr>
<td>5:1+Join(2)</td>
<td>0.001s</td>
<td>0.05s</td>
<td>0.004s</td>
<td>0.01s</td>
<td>0.02s</td>
<td>0.02s</td>
</tr>
<tr>
<td>6:5+Join(3)</td>
<td>0.001s</td>
<td>0.05s</td>
<td>0.004s</td>
<td>0.01s</td>
<td>0.02s</td>
<td>0.02s</td>
</tr>
</tbody>
</table>

N=Node, S=Sequence, and L=Loop.

From Table 8.3 and 8.3, we can see that on average query 2 and query 5 take more execution time than other three queries. This is because query 2 and query 5 have more hits in both collections than other queries, as shown in Table 8.3. We can also see that query 2 takes more time than query 5, though query 5 has more hits. This is because query 2 has a wildcard node, which matches with all node features. Therefore it is logical that a query with a wildcard node takes more time because more comparisons are required.

### Table 8.4: Execution Time Using 6040 Synthetic Process Models

<table>
<thead>
<tr>
<th>Features(size)</th>
<th>query1</th>
<th>query2</th>
<th>query3</th>
<th>query4</th>
<th>query5</th>
<th>$T_{avg}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:N(0)+S(1)+L(1)</td>
<td>0.004s</td>
<td>0.28s</td>
<td>0.02s</td>
<td>0.03s</td>
<td>0.19s</td>
<td>0.10s</td>
</tr>
<tr>
<td>2:1+S(2)+L(2)</td>
<td>0.003s</td>
<td>0.28s</td>
<td>0.02s</td>
<td>0.03s</td>
<td>0.19s</td>
<td>0.10s</td>
</tr>
<tr>
<td>3:1+Split(2)</td>
<td>0.004s</td>
<td>0.28s</td>
<td>0.02s</td>
<td>0.03s</td>
<td>0.18s</td>
<td>0.10s</td>
</tr>
<tr>
<td>4:3+Split(3)</td>
<td>0.003s</td>
<td>0.28s</td>
<td>0.02s</td>
<td>0.03s</td>
<td>0.19s</td>
<td>0.10s</td>
</tr>
<tr>
<td>5:1+Join(2)</td>
<td>0.003s</td>
<td>0.28s</td>
<td>0.02s</td>
<td>0.03s</td>
<td>0.19s</td>
<td>0.10s</td>
</tr>
<tr>
<td>6:5+Join(3)</td>
<td>0.004s</td>
<td>0.28s</td>
<td>0.02s</td>
<td>0.03s</td>
<td>0.19s</td>
<td>0.10s</td>
</tr>
</tbody>
</table>

N=Node, S=Sequence, and L=Loop.

8.4 Evaluation for Process Similarity Search

This section presents the evaluations of the process similarity technique described in this thesis. The evaluations determine the execution time of the algorithm and the quality of the search results that it returns. The experiments were run with
Table 8.5: Properties of Querying with Synthetic models

<table>
<thead>
<tr>
<th>Query Name</th>
<th>Hits in 604 Models</th>
<th>Hits in 6040 Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query 1</td>
<td>1</td>
<td>21</td>
</tr>
<tr>
<td>Query 2</td>
<td>14</td>
<td>166</td>
</tr>
<tr>
<td>Query 3</td>
<td>4</td>
<td>43</td>
</tr>
<tr>
<td>Query 4</td>
<td>2</td>
<td>34</td>
</tr>
<tr>
<td>Query 5</td>
<td>34</td>
<td>343</td>
</tr>
</tbody>
</table>

both real-life and synthetic process models. For the real-life process models, one homogeneous evaluation was performed in which the search models were taken from the collection that was searched and one heterogeneous evaluation was performed in which the search models were taken from a different model collection. For the synthetic process models, search models were run with a collection of hundreds of process models and a collection of thousands of process models. We use Definition 6.7 to match nodes for process similarity in the experiments.

8.4.1 Homogeneous Real-life Process Models

In this subsection, we present the homogeneous evaluation with real-life process models. We first explain the setup of the evaluation and then the results.

Evaluation Setup

We have two experimental setups: one for evaluating the quality of retrieved results and one for evaluating the execution time, respectively.

Both experiments are performed on the collection of SAP reference models [27]. To evaluate the quality of retrieved results, we use the same evaluation collection as in [30]. This collection consists of 100 process models that were extracted from the collection of SAP reference models. In addition to that we extracted 10 process models as search models. Consequently, there are 1000 combinations of a search model and a model in the collection for which the similarity can be determined. For each of those combinations three human observers judged whether the process model is a relevant search result for a particular search model. Therefore, we can determine the quality of the search results that are returned by a particular
8.4. Evaluation for Process Similarity Search

algorithm by comparing them to the relevance judgement that is given by the human observers. We can quantify the quality in terms of the R-Precision [23].

Definition 8.15 (R-Precision). Let $D$ be the set of process models, $Q$ be the set of search models and relevant : $Q \rightarrow \mathbb{P}(D)$ be the function that returns the set of relevant process models for each search model (as determined by the human observer). Given the list of search results $D = [d_1, d_2, ..., d_n]$ for a query $q$ with $d_i \in D$, the R-Precision is the precision of the first $R$ results, where $R = |\text{relevant}(q)|$ is the total number of process models that is relevant to the query:

$$\text{R-Precision} = \frac{|\{d_i \in D | i \leq n, i \leq R, d_i \in \text{relevant}(q)\}|}{R}$$

We compare the R-Precision of the greedy algorithm that we developed in previous work [30] to the R-Precision of the improved greedy algorithm that is described in this thesis. We use the greedy algorithm, because it is the fastest algorithm of the ones we studied [30] and, therefore, provides a lower-bound for improvements in execution time.

To evaluate the execution time, we compare the 10 search models with all 604 business process models in the collection of SAP reference models, instead of just the 100 process models. We do this, because to compute the execution time we do not need the human judgement and computing the execution time for a larger set of models leads to a more realistic result. We record the average execution time per query. Further more, we randomly select 10 models from synthetic models and compare them with 604 and 6040 synthetic models respectively.

Evaluation Results

Table 8.6 shows the quality of the results that are retrieved by the greedy algorithm and those returned by the improved greedy algorithm in combination with similarity estimation, based on various feature types.

The rows in the table show the features that are used to do the feature-based similarity estimation. In the first row no feature-based similarity estimation is done. This row lists the performance of the greedy algorithm. In the second row similarity estimation is done based only on node features (of size 0). In the third row similarity estimation is done based on node features plus sequence features of size 1 and so on.
Table 8.6: Result Quality of the Homogeneous Evaluation

<table>
<thead>
<tr>
<th>Feature(n)</th>
<th>Occurrence</th>
<th>Matches</th>
<th>Rel</th>
<th>PoR</th>
<th>Ir</th>
<th>R-Prec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Work [30]</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0.84</td>
</tr>
<tr>
<td>1:Node(0)</td>
<td>374</td>
<td>581</td>
<td>5.5</td>
<td>10.9</td>
<td>83.6</td>
<td>0.84</td>
</tr>
<tr>
<td>2:1+Seq(1)</td>
<td>+267</td>
<td>+197</td>
<td>8.1</td>
<td>8</td>
<td>83.9</td>
<td>0.83</td>
</tr>
<tr>
<td>3:2+Seq(2)</td>
<td>+175</td>
<td>+96</td>
<td>7.8</td>
<td>10.1</td>
<td>82.1</td>
<td>0.83</td>
</tr>
<tr>
<td>4:2+Split(2)</td>
<td>+87</td>
<td>+93</td>
<td>7.8</td>
<td>10.1</td>
<td>82.1</td>
<td>0.83</td>
</tr>
<tr>
<td>5:4+Split(3)</td>
<td>+23</td>
<td>+11</td>
<td>7.8</td>
<td>10.1</td>
<td>82.1</td>
<td>0.83</td>
</tr>
<tr>
<td>6:2+Join(2)</td>
<td>+58</td>
<td>+18</td>
<td>7.8</td>
<td>10.1</td>
<td>82.1</td>
<td>0.83</td>
</tr>
<tr>
<td>7:6+Join(3)</td>
<td>+14</td>
<td>+1</td>
<td>7.8</td>
<td>10.1</td>
<td>82.1</td>
<td>0.83</td>
</tr>
</tbody>
</table>

The columns in the table show the properties of the features and similarity estimation based on the features. First, they show the number of times features of a given type occur in the set of process models and the number of times features of a certain type match in the set of process models. For example, in the set of process models, there could be four nodes labeled ‘A’. These nodes count as four occurrences of the node feature type. Because of their high label feature similarity, these nodes can be considered to match. This leads to six matches, because each of the four nodes can be matches to each of the others. Second, the columns show the average number of process models that, after the similarity estimation step, are estimated as being relevant (Rel), potentially relevant (PoR) and irrelevant (Ir) over the ten search executions. Third, the columns show the average R-Precision (R-Prec) over the ten search executions.

The table shows that when similarity estimation is done based only on node features on average 5.5 models are estimated to be relevant, 10.9 models to be potentially relevant and 83.6 models as irrelevant. Therefore, in this situation, the improved greedy algorithm only has to be used to measure the similarity of about 11% of the total number of process models, about 6% of the models are immediately judged as relevant and the remaining models are judged as irrelevant. In this case the quality of the returned results in terms of R-Precision remains the same. If sequences of size two are also used to perform the similarity estimation, only 8% of the process models has to be compared using the improved greedy algorithm. However, this does lead to a slightly lower R-Precision. Inclusion of other types of
features does not improve the similarity estimation any further.

Table 8.7 shows the execution time of the similarity search both when only using the greedy algorithm and when using certain feature types for similarity estimation and the improved greedy algorithm.

Table 8.7: Execution Time of the Homogeneous Evaluation with 604 SAP Reference Models

<table>
<thead>
<tr>
<th>Features(n)</th>
<th>Rel</th>
<th>PoR</th>
<th>Ir</th>
<th>$T_{est}$</th>
<th>$T_{com}$</th>
<th>$T_{avg}^{total}$</th>
<th>$T_{min}^{total}$</th>
<th>$T_{max}^{total}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Work [30]</td>
<td>0</td>
<td>604</td>
<td>0</td>
<td>0.00s</td>
<td>0.60s</td>
<td>0.60s</td>
<td>0.16s</td>
<td>1.45s</td>
</tr>
<tr>
<td>1:Node(0)</td>
<td>7</td>
<td>73</td>
<td>524</td>
<td>0.05s</td>
<td>0.04s</td>
<td>0.09s</td>
<td>0.03s</td>
<td>0.14s</td>
</tr>
<tr>
<td>2:1+Seq(1)</td>
<td>13.7</td>
<td>44.9</td>
<td>554.4</td>
<td>0.05s</td>
<td>0.02s</td>
<td>0.07s</td>
<td>0.03s</td>
<td>0.09s</td>
</tr>
<tr>
<td>3:2+Seq(2)</td>
<td>9.5</td>
<td>73.2</td>
<td>521.3</td>
<td>0.05s</td>
<td>0.05s</td>
<td>0.10s</td>
<td>0.03s</td>
<td>0.15s</td>
</tr>
<tr>
<td>4:2+Split(2)</td>
<td>9.5</td>
<td>73.2</td>
<td>521.3</td>
<td>0.05s</td>
<td>0.05s</td>
<td>0.10s</td>
<td>0.03s</td>
<td>0.15s</td>
</tr>
<tr>
<td>5:4+Split(3)</td>
<td>9.5</td>
<td>73.2</td>
<td>521.3</td>
<td>0.05s</td>
<td>0.05s</td>
<td>0.10s</td>
<td>0.03s</td>
<td>0.15s</td>
</tr>
<tr>
<td>6:2+Join(2)</td>
<td>9.5</td>
<td>73.2</td>
<td>521.3</td>
<td>0.05s</td>
<td>0.05s</td>
<td>0.10s</td>
<td>0.03s</td>
<td>0.15s</td>
</tr>
<tr>
<td>7:6+Join(3)</td>
<td>9.5</td>
<td>73.2</td>
<td>521.3</td>
<td>0.05s</td>
<td>0.05s</td>
<td>0.10s</td>
<td>0.03s</td>
<td>0.15s</td>
</tr>
</tbody>
</table>

The execution time consists of two parts: the time it takes to estimate the similarity and classify process models as relevant (Rel), potentially relevant (PoR) or irrelevant (Ir), denoted $T_{est}$; and the time it takes to compute the similarity for the models classified as potentially relevant, denoted $T_{com}$. Table 8.7 shows the average estimation and execution times over the ten search models. In addition to that it shows the average total time over the ten search models and the (minimum) time of processing the search that takes the least time and the (maximum) time of processing the search that takes the most time.

The table shows that, on average, estimating similarity based on node features helps to retrieve similar models 6.7 times faster and in Table 8.6 we can see that this does not impact the quality of the search results. Also including sequence features of size two helps retrieve similar models 8.6 times faster, but in Table 8.6 we can see that this reduces the quality of the results by about 0.01 in terms of R-Precision as a tradeoff. However, including more features increases the execution time, and in Table 8.6 we can see that the quality remains the same as using node features and sequence features of size two.

Fig. 8.5 summarizes the results of the homogeneous evaluation running with SAP Reference Models, by showing both the quality and the execution time evaluation.
We can see that the algorithm in this thesis reduces the execution time (from 0.6s to less than 0.1s per search on average) with a stable quality (0.83-0.84 on average) in terms of R-Precision.

![Figure 8.5: Results of the Homogeneous Evaluation with SAP Reference Models](image)

Similarity estimation depends on the following parameters:

- $dcutoff$, which is a parameter that determines whether a role feature is considered to be discriminative (Definition 6.5).
- $lcutoff_{\text{high}}$, $rcutoff$ and $lcutoff_{\text{med}}$, which are parameters that determine what is considered to be a sufficiently similar for a feature to match (Definition 6.7).
- $ratio_r$ and $ratio_p$, which are parameters that determine which class a process model belongs to based on the fraction of features that match with the search model (Definition 6.15).

We vary each of these parameters from 0 to 1 in increments of 0.1 and ran the experiments with all possible combinations of parameter values within this range. We use the parameters that, on average, give the highest R-Precision or the fewest potentially relevant models with respect to the search to show attractive tradeoffs. The values that we use are $dcutoff = 0.3$, $lcutoff_{\text{high}} = 0.8$, $rcutoff = 1.0$ and $lcutoff_{\text{med}} = 0.2$. The other two parameters also depend on the type of features we use. For the node feature (the second row in Table 8.6 or 8.7), $ratio_r = 0.5$; otherwise, $ratio_r = 0.2$. For the node and sequence (with two nodes) features (the second and third rows in Table 8.6 or 8.7), $ratio_p = 0.1$; otherwise, $ratio_p = 0.0$.

The improved greedy algorithm for process similarity search depends on the following parameters:

- $wskipn$, $wskipe$ and $wsubn$, which denote the weights given to node deletion, node substitution and edge deletion (Definition 6.16).
8.4. Evaluation for Process Similarity Search

- $k$, which denotes how many most similar nodes are considered for a search node (Definition 6.17).
- $w_l$ and $w_r$, which denote the weights given to the label and role similarities (Definition 6.18).

For the first group, we used the same values as in [30], i.e., $w_{skipn} = 0.1$, $w_{skipe} = 0.4$ and $w_{subn} = 0.9$. For the second group, we varied each of these parameters from 1 to 10 in increments of 1, and it returns best results when $k = 3$. For the third group, we varied each of these parameters from 0 to 1 in increments of 0.1 and ran the experiments with all possible combinations of parameter values within this range. We used the parameters that give best results, i.e., $w_l = 1.0$ and $w_r = 0.6$.

Note that the parameter settings are specifically tuned to give the best results for this dataset. Parameter values that are generically applicable should be obtained through additional experiments on other process model collections. However, note that a change in parameter settings should not change the conclusions about the comparison between the greedy algorithm and the algorithm in this thesis, because both algorithm profit equally from the optimization of the parameters for the evaluation dataset.

8.4.2 Heterogeneous Real-life Process Models

In this subsection, we present the heterogeneous evaluation with real-life process models. We first explain the setup of the evaluation and then the results.

Evaluation Setup

In the heterogeneous evaluation, the model collection was extracted from the same collection as for the homogeneous evaluation. However, the search models were taken from a different collection of business process models, which represent the processes of a large manufacturing company. Ten search models were extracted.

The main difference between the heterogeneous and the homogeneous evaluation is that, in the homogeneous evaluation, it is more clear which models are similar to a given model. For example, the SAP Reference Model contains 7 purchasing models that resemble each other strongly. Consequently, given one of the purchasing models, it is very easy to find the other, similar, ones. For the heterogeneous
evaluation, this is more difficult: given a purchasing model (that is not from the SAP Reference Model), similar models are less easy to identify. Therefore, the similarity estimation step will initially lead to more models that are potentially relevant. Consequently, we expect that the similarity estimation step will lead to a smaller efficiency improvement.

Both the SAP reference models and the manufacturing models cover a large number of business functions, which are distributed over different “branches” of the business process model collections (e.g., the collection of SAP reference models has 29 branches and a total of 604 business process models). Models that belong to different branches are typically not similar. To develop a collection of business process models that can be feasibly compared by human observers and that also contains models that are similar, selections from both model collections were made, by first determining similar branches and then selecting models from similar branches. As shown in Table 8.8, 10 manufacturing models are selected as search models, and 97 SAP reference models are selected as models in dataset, taking overlapping or related branches from the models. The metrics for evaluating quality and time are the same as the previous evaluation, i.e., the R-Precision and the average execution time per search.

<table>
<thead>
<tr>
<th>Branch/Business Function</th>
<th>nr. of search models</th>
<th>nr. of document models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procurement</td>
<td>3</td>
<td>37</td>
</tr>
<tr>
<td>Delivery and Invoicing</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Production planning</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>Sales</td>
<td>4</td>
<td>43</td>
</tr>
<tr>
<td>Business Planning</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Management</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Evaluation Results**

Table 8.9 shows the results of the greedy algorithm and the algorithm in this thesis. The experiments are run on the same computer as the homogeneous evaluation. The rows show the features that are used to estimate the similarity. The columns show the classification of models, result quality and the execution times per search.
Process Models are classified as relevant (Rel), potentially relevant (PoR) or irrelevant (Ir) models. Result quality is also measured by R-Precision (R-Prec). The average execution time \( T_{\text{avg}}^{\text{total}} \) consists of the estimation time based on features \( T_{\text{est}} \) and the computation time by the improved greedy algorithm \( T_{\text{com}} \). Besides these the columns also show the execution time for the search that take least \( T_{\text{min}}^{\text{total}} \) and most \( T_{\text{max}}^{\text{total}} \) time.

The table shows that, by using node features only, around 20% of process models need to be checked with the improved greedy algorithm; the execution time is reduced by 8 times; while the quality is reduced by 0.02 in terms of R-Precision. These findings support our expectation that in the heterogeneous case more models will need to be checked with the greedy algorithm than in the homogeneous case (in the homogeneous case 10% of the process models need to be checked). By further including sequence with two nodes features, around 12% of process models need to be checked with the improved greedy algorithm; the execution time is reduced by 10.7 times; while the quality is reduced by 0.04 in terms of R-Precision. Similar to the previous evaluation, the results do not improve anymore by including more features.

Fig. 8.6 summarizes the results of the heterogeneous evaluation, by showing both the quality and the execution time evaluation. We can see that the algorithm in this thesis significantly reduces the execution time with a small decrease in quality.

For the heterogeneous evaluation we changed the values for the parameters \( lcutoff_{\text{high}} = 0.2, lcutoff_{\text{med}} = 0.1, wskipe = 0.0 \) and \( wsubn = 0.1 \) to obtain the best results. The values of other parameters stay the same as for the homogeneous evaluation.
Table 8.9: Results of the Heterogeneous Evaluation

<table>
<thead>
<tr>
<th>Features(n)</th>
<th>Rel</th>
<th>PoR</th>
<th>Ir</th>
<th>R-Prec</th>
<th>$T_{est}$</th>
<th>$T_{com}$</th>
<th>$T_{avg}^{total}$</th>
<th>$T_{min}^{total}$</th>
<th>$T_{max}^{total}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous [30]</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0.56</td>
<td>0.00s</td>
<td>0.32s</td>
<td>0.32s</td>
<td>0.20s</td>
<td>0.51s</td>
</tr>
<tr>
<td>1:Node(0)</td>
<td>74.2</td>
<td>20.4</td>
<td>5.4</td>
<td>0.54</td>
<td>0.02s</td>
<td>0.02s</td>
<td>0.04s</td>
<td>0.02s</td>
<td>0.06s</td>
</tr>
<tr>
<td>2:1+Seq(1)</td>
<td>83</td>
<td>11.6</td>
<td>5.4</td>
<td>0.52</td>
<td>0.02s</td>
<td>0.01s</td>
<td>0.03s</td>
<td>0.02s</td>
<td>0.05s</td>
</tr>
<tr>
<td>3:2+Seq(2)</td>
<td>85</td>
<td>9.6</td>
<td>5.4</td>
<td>0.50</td>
<td>0.02s</td>
<td>0.01s</td>
<td>0.03s</td>
<td>0.02s</td>
<td>0.05s</td>
</tr>
<tr>
<td>4:2+Split(2)</td>
<td>85</td>
<td>9.6</td>
<td>5.4</td>
<td>0.50</td>
<td>0.02s</td>
<td>0.01s</td>
<td>0.03s</td>
<td>0.02s</td>
<td>0.05s</td>
</tr>
<tr>
<td>5:4+Split(3)</td>
<td>85</td>
<td>9.6</td>
<td>5.4</td>
<td>0.50</td>
<td>0.02s</td>
<td>0.01s</td>
<td>0.03s</td>
<td>0.02s</td>
<td>0.05s</td>
</tr>
<tr>
<td>6:2+Join(2)</td>
<td>85</td>
<td>9.6</td>
<td>5.4</td>
<td>0.50</td>
<td>0.02s</td>
<td>0.01s</td>
<td>0.03s</td>
<td>0.02s</td>
<td>0.05s</td>
</tr>
<tr>
<td>7:6+Join(3)</td>
<td>85</td>
<td>9.6</td>
<td>5.4</td>
<td>0.50</td>
<td>0.02s</td>
<td>0.01s</td>
<td>0.03s</td>
<td>0.02s</td>
<td>0.05s</td>
</tr>
</tbody>
</table>

The quality of the homogeneous experiment is higher (0.84 v.s. 0.54). This is because similar tasks are typically labeled with the same terms in the same collection, but with different terms in different collections. Therefore, it is easier to establish task similarity based on label similarity in homogeneous datasets. Moreover, we currently use string edit distance to compute label similarity in this thesis, which is a naive label similarity metric that cannot deal well with different (synonymous) terms being used in similar labels. The result quality of the heterogeneous experiment should therefore be improved by considering synonyms [31, 34] and domain ontologies [37]. We can also see that the execution time of the heterogeneous experiment is much less (0.60s v.s. 0.32s). This is because the size of search models. Although the average sizes are almost the same for both sets of search models (20.7 v.s. 20.3), the maximal size differs a lot (130 v.s. 35). Consequently, the slowest search of the homogeneous experiment takes 1.45s, while the slowest search of the heterogeneous experiment only takes 0.51s. This causes the homogeneous experiment to take more time on average.

### 8.4.3 Synthetic Process Models

In this subsection, we present the evaluation with synthetic process models. We first explain the setup of the evaluation and then the results.
8.4. Evaluation for Process Similarity Search

Evaluation Setup

To investigate the execution time of the technique when the size of collections increases, we generated 6040 synthetic process models using the generator described in Section 8.2. We performed two experiments, running ten queries with a collection of 604 synthetic models and a collection of 6040 synthetic models. The queries were randomly selected from the collection of 604 synthetic process models. The results of the former experiment are used to compare with the results of the experiment with 604 SAP reference models, which evaluates whether the process similarity search technique in this thesis works well with different collections. The results of the latter experiment are used to compare with the results of the former one, which evaluates the scalability of the process similarity search technique in this thesis.

Evaluation Results

Table 8.10 and 8.11 show the execution time of the similarity search running with 604 and 6040 synthetic models respectively. The structure of the tables is the same as Table 8.7. In all these experiments the technique in this thesis also performs best by including node features and sequence features of size two. Table 8.10 shows that, on average, estimating similarity based on node features and sequence features of size two helps to retrieve similar models 12 times faster. Table 8.11 shows that, on average, estimating similarity based on node features and sequence features of size 1 helps to retrieve similar models 36.8 times faster. Table 8.11 also shows that, in the worst case, the total search time for the greedy algorithm [30] is already 10.97 seconds, which is slower than the response time that one would expect of a search engine; while in worst case, the total search time for the technique in this thesis is still within milliseconds using node features (0.3s) or node features and sequence features of size 1 (0.6s).

By comparing Table 8.10 and 8.11 we can see that the average execution time running with the 604 SAP reference models and the 604 synthetic models are different. This is expected for the following two reasons. Firstly, the properties of the process models in these two collections are different, i.e., number of features and occurrences of a feature. As a result, the number of PoR models are different, and $T_{\text{com}}$ is linear with respect to the number of PoR models. The number of PoR models in the experiment with the SAP reference models is 2 to 4 times more than that.
Table 8.10: Execution Time Running with 604 Synthetic Models

<table>
<thead>
<tr>
<th>Features(n)</th>
<th>Rel</th>
<th>PoR</th>
<th>Ir</th>
<th>$T_{est}$</th>
<th>$T_{com}$</th>
<th>$T_{avg\ total}$</th>
<th>$T_{min\ total}$</th>
<th>$T_{max\ total}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Work</td>
<td>0</td>
<td>604</td>
<td>0</td>
<td>0.00s</td>
<td>0.48s</td>
<td>0.48s</td>
<td>0.13s</td>
<td>1.13s</td>
</tr>
<tr>
<td>1:Node(0)</td>
<td>1.4</td>
<td>19.3</td>
<td>583.3</td>
<td>0.03s</td>
<td>0.007s</td>
<td>0.04s</td>
<td>0.001s</td>
<td>0.10s</td>
</tr>
<tr>
<td>2:1+Seq(1)</td>
<td>5.0</td>
<td>16.9</td>
<td>552.1</td>
<td>0.03s</td>
<td>0.004s</td>
<td>0.03s</td>
<td>0.001s</td>
<td>0.10s</td>
</tr>
<tr>
<td>3:2+Seq(2)</td>
<td>3.3</td>
<td>33.4</td>
<td>567.3</td>
<td>0.03s</td>
<td>0.02s</td>
<td>0.05s</td>
<td>0.001s</td>
<td>0.20s</td>
</tr>
<tr>
<td>4:2+Split(2)</td>
<td>3.3</td>
<td>33.4</td>
<td>567.3</td>
<td>0.03s</td>
<td>0.02s</td>
<td>0.05s</td>
<td>0.001s</td>
<td>0.20s</td>
</tr>
<tr>
<td>5:4+Split(3)</td>
<td>3.3</td>
<td>33.4</td>
<td>567.3</td>
<td>0.03s</td>
<td>0.02s</td>
<td>0.05s</td>
<td>0.001s</td>
<td>0.20s</td>
</tr>
<tr>
<td>6:2+Join(2)</td>
<td>3.3</td>
<td>33.4</td>
<td>567.3</td>
<td>0.03s</td>
<td>0.02s</td>
<td>0.05s</td>
<td>0.001s</td>
<td>0.20s</td>
</tr>
<tr>
<td>7:6+Join(3)</td>
<td>3.3</td>
<td>33.4</td>
<td>567.3</td>
<td>0.03s</td>
<td>0.02s</td>
<td>0.05s</td>
<td>0.001s</td>
<td>0.20s</td>
</tr>
</tbody>
</table>

Table 8.11: Execution Time Running with 6040 Synthetic Models

<table>
<thead>
<tr>
<th>Features(n)</th>
<th>Rel</th>
<th>PoR</th>
<th>Ir</th>
<th>$T_{est}$</th>
<th>$T_{com}$</th>
<th>$T_{avg\ total}$</th>
<th>$T_{min\ total}$</th>
<th>$T_{max\ total}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Work</td>
<td>0</td>
<td>6040</td>
<td>0</td>
<td>0.00s</td>
<td>4.78s</td>
<td>4.78s</td>
<td>1.36s</td>
<td>10.97s</td>
</tr>
<tr>
<td>1:Node(0)</td>
<td>6.8</td>
<td>175.6</td>
<td>5857.6</td>
<td>0.10s</td>
<td>0.05s</td>
<td>0.16s</td>
<td>0.003s</td>
<td>0.30s</td>
</tr>
<tr>
<td>2:1+Seq(1)</td>
<td>48.5</td>
<td>108.2</td>
<td>5883.3</td>
<td>0.11s</td>
<td>0.03s</td>
<td>0.13s</td>
<td>0.003s</td>
<td>0.60s</td>
</tr>
<tr>
<td>3:2+Seq(2)</td>
<td>28.2</td>
<td>315.8</td>
<td>5696.0</td>
<td>0.10s</td>
<td>0.18s</td>
<td>0.29s</td>
<td>0.003s</td>
<td>1.00s</td>
</tr>
<tr>
<td>4:2+Split(2)</td>
<td>28.2</td>
<td>315.8</td>
<td>5696.0</td>
<td>0.11s</td>
<td>0.19s</td>
<td>0.29s</td>
<td>0.003s</td>
<td>1.00s</td>
</tr>
<tr>
<td>5:4+Split(1)</td>
<td>28.2</td>
<td>315.8</td>
<td>5696.0</td>
<td>0.11s</td>
<td>0.19s</td>
<td>0.30s</td>
<td>0.003s</td>
<td>1.00s</td>
</tr>
<tr>
<td>6:2+Join(2)</td>
<td>28.5</td>
<td>315.5</td>
<td>5696.0</td>
<td>0.11s</td>
<td>0.18s</td>
<td>0.29s</td>
<td>0.003s</td>
<td>1.00s</td>
</tr>
<tr>
<td>7:6+Join(3)</td>
<td>28.5</td>
<td>315.5</td>
<td>5696.0</td>
<td>0.11s</td>
<td>0.19s</td>
<td>0.29s</td>
<td>0.003s</td>
<td>1.00s</td>
</tr>
</tbody>
</table>

in the experiment with the synthetic models. Therefore, it is reasonable that the experiment with the SAP reference models takes more execution time. Secondly, both groups of search models are randomly selected in these two experiments. In the above experiments with the SAP reference models and the synthetic models, the average size of search models are 4.0 and 3.7 respectively (without considering events and gateways of EPCs). Therefore, the average size of search models could be different. The greater the average size of search models is, the bigger $T_{avg\ total}$ is.

The comparison time between a search model and PoR models is linear to the number of PoR models. Therefore, the numbers of PoR models can be used to analyze the results. Firstly, we use the number of PoR models in the 604 synthetic model experiment, the number of PoR models of the 604 SAP model experiment,
and the comparison time between search models and PoR models of the 604 synthetic model experiment to calculate the comparison time between search models and PoR models of the 604 SAP model experiment; then check whether the calculation time is close to the real time. If it is, we can use the the same method to calculate the comparison time for 6040 SAP reference model if there were such a collection. This provides an indication for the execution time for searching large collections of real-life process models.

Table 8.12 shows the analysis results. The factors and formulas used to do the analysis in explained in the following list.

- **Factor0** is a ‘comparison’ ratio between the execution time of running the greedy algorithm [30] with the 604 SAP reference models and the 604 synthetic models. Formally, Factor0\(=\frac{T_{\text{com}}^{\text{SAP604}}}{T_{\text{com}}^{\text{Syn604}}}=0.60s/0.48s=1.25\).

- **Factor1** is a ‘PoR’ ratio between the numbers of PoR models in the experiments with the 604 SAP reference models (\(\text{PoR}_{\text{SAP604}}\)) and with the 604 synthetic models (\(\text{PoR}_{\text{Syn604}}\)). Formally, Factor1\(=\frac{\text{PoR}_{\text{SAP604}}}{\text{PoR}_{\text{Syn604}}}\).

- **ET\(_{\text{com}}^{\text{SAP604}}\)** is a calculation of the time for comparing search models with PoR models in the experiment with the 604 SAP reference models based on the comparison time with the 604 synthetic models (\(T_{\text{com}}^{\text{Syn604}}\)), the ‘comparison’ ratio (Factor0), and the ‘PoR’ ratio (Factor1). Formally, \(ET_{\text{com}}^{\text{SAP604}}=T_{\text{com}}^{\text{Syn604}} \times \text{Factor0} \times \text{Factor1}\).

- **\(T_{\text{com}}^{\text{SAP604}}\)** is the real execution time for comparing search models with PoR models in the experiment with the 604 SAP reference models (from Table 8.7).

- **ET\(_{\text{com}}^{\text{SAP6040}}\)** is a calculation of the time for comparing search models with PoR models in the experiment with 6040 SAP reference models based on the comparison time in the experiment with the 6040 synthetic models (ET\(_{\text{com}}^{\text{Syn6040}}\)), the ‘comparison’ ratio (Factor0), and the ‘PoR’ ratio (Factor1). Formally, \(ET_{\text{com}}^{\text{SAP6040}}=T_{\text{com}}^{\text{Syn6040}} \times \text{Factor0} \times \text{Factor1}\).

The columns \(ET_{\text{com}}^{\text{SAP604}}\) and \(T_{\text{com}}^{\text{SAP604}}\) of the table show that the calculation time and the real time, comparing search models with PoR models in the experiment with the 604 SAP reference models, are close. The columns ET\(_{\text{com}}^{\text{SAP6040}}\) shows that, by calculation, the estimation time, comparing search models with PoR models of 6040 SAP reference models, is still within milliseconds.
Table 8.12: Execution Time Estimation

<table>
<thead>
<tr>
<th>Features</th>
<th>Factor1</th>
<th>ET_{SAP604}^{com}</th>
<th>T_{SAP604}^{com}</th>
<th>ET_{SAP6040}^{com}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:Node(0)</td>
<td>3.78</td>
<td>0.03s</td>
<td>0.04s</td>
<td>0.24s</td>
</tr>
<tr>
<td>2:1+Seq(1)</td>
<td>2.66</td>
<td>0.01s</td>
<td>0.02s</td>
<td>0.10s</td>
</tr>
<tr>
<td>3:2+Seq(2)</td>
<td>2.19</td>
<td>0.05s</td>
<td>0.05s</td>
<td>0.49s</td>
</tr>
<tr>
<td>4:2+Split(2)</td>
<td>2.19</td>
<td>0.05s</td>
<td>0.05s</td>
<td>0.52s</td>
</tr>
<tr>
<td>5:4+Split(3)</td>
<td>2.19</td>
<td>0.05s</td>
<td>0.05s</td>
<td>0.52s</td>
</tr>
<tr>
<td>6:2+Join(2)</td>
<td>2.19</td>
<td>0.05s</td>
<td>0.05s</td>
<td>0.49s</td>
</tr>
<tr>
<td>7:6+Join(3)</td>
<td>2.19</td>
<td>0.05s</td>
<td>0.05s</td>
<td>0.49s</td>
</tr>
</tbody>
</table>

8.5 conclusion

We have applied the proposed process retrieval techniques to both real-life and synthetic collections. The sizes of the collections are 604 and 6040. We have seen that the techniques outperform existing techniques by at least one order of magnitude. In the experiments, our techniques always retrieve results for a query or search model within milliseconds, even in the worst cases. This has important practical implications, because process retrieval without efficiency improving can take more than 10 seconds on a real-size (hundreds or thousands) collection. This is not acceptable to computer users these days, who are used to retrieve results within milliseconds, e.g., Google.
Part IV

Closure
Chapter 9

Conclusions

9.1 Introduction

The research goal of this thesis is to enhance BP Model Repository technology for large business model collections in a relevant aspect. This chapter discusses whether the research goal is met by the techniques proposed in this thesis. The remainder of the chapter is organized as follows. Sections 9.2 and 9.3 present the contributions and limitations of the techniques proposed in this thesis. Section 9.4 analyzes the research approach. Section 9.5 provides directions to improve the techniques.

9.2 Contributions

This section first presents the contributions of the techniques presented in this thesis. The contributions relate to the deliverables created by performing the research steps defined in Chapter 1.

The first contribution of this thesis is the state of the art analysis of BP Model Repositories. This analysis is performed as follows.

Chapter 2 presents a descriptive framework consisting of a management model and a reference architecture by observing existing BP Model Repositories. The management model consists of three sub-models, i.e., process data model, process function model, and process management model. The process data model describes data, e.g., activities and control-flows, stored in the repositories that can repre-
sent processes; the process function and management models describe functions, e.g., process retrieval, and management, e.g., process version management, which operate and manage process models and related data stored in the process data model. The reference architecture divides functionality of BP Model Repositories, described in the management model, into a four-layer architecture, i.e., the storage layer, the database management layer, the process repository management layer, and the presentation layer. The framework serves as a guide for the development of BP Model Repositories. It also serves as a basis to evaluate the completeness of a BP Model Repository.

Chapter 3 presents a comparison between existing BP Model Repositories based on the proposed framework in Chapter 2. We check which functionality described in the framework is implemented by which existing BP Model Repositories. The comparison shows that BP Model Repositories with complete functionality according to the framework are not yet available. In particular, the following functionality is currently missing in BP Model Repositories: no process specific index is provided by existing BP Model Repositories; process collaboration, administration and monitoring tools and report generators are rarely integrated into BP Model Repositories; and process view, configuration, and lifecycle management is rarely supported. Therefore, there is room to enhance BP Model Repository technology.

After the analysis of functionality currently provided by BP Model Repositories, we selected a particular function to improve on. This is the indexing function, which enables more efficient process retrieval. Therefore, the second contribution of this thesis is the enhancement of the efficiency of process retrieval. Process retrieval techniques that are more efficient are developed in the following steps.

To better understand the state-of-the-art of the process retrieval technology, Chapter 4 presents a survey, which covers both process querying and similarity search techniques. Most importantly, we check whether these techniques provide process specific indexes. The result shows that two of the techniques provide an index. However, no index is provided to support both process querying and similarity search. Therefore, we draw the conclusion that providing such an index enhances the process retrieval technology.

Chapter 5 presents how to construct a feature-based index, called PFIndex, to enhance the efficiency of process querying and similarity search. Comparing two process models (graphs) is an NP-Complete problem and requires considerable
9.2. Contributions

computation time. Therefore, small and representative fragments of process models are used as features, which are organized hierarchically based on the parent-child relations between features in a PFIndex, as defined in Chapter 5. As such, we compare fragments of process models instead of process models to estimate whether a query or a search model is matching with a process model, which should be more efficient than comparing two process models directly. A PFIndex can work together with process retrieval techniques to enhance their efficiency.

Chapter 6 proposes process retrieval techniques that are integrated with a PFIndex. For process querying, common features between a query model and models in a collection are determined efficiently using the PFIndex. A process model is a candidate for matching with a given query model, if it contains matches for all basic query features of the query model. Then further checks for contradictions and advanced query features are performed based on matching features to decide whether a candidate satisfy the query. For process similarity search, common features between a query model and models in a collection are again determined efficiently using the PFIndex. The similarity of a search model and a process model is estimated by the ratio of matching features. Models that are relevant, potentially relevant, or irrelevant to a search model are determined, based on their ratios of matching features with respect to the search model. If the process model is potentially relevant with respect to the query model, an algorithm, the improved greedy algorithm, is performed to compute the similarity of these two models.

Chapter 7 presents an implementation of proposed process retrieval techniques. The architecture of the implementation is based on the reference architecture proposed in Chapter 2. The implementation provides the functionality as the proposed techniques are expected to provide, and can be used to evaluate the efficiency of proposed techniques.

Chapter 8 presents experiments to evaluate the proposed process retrieval techniques through the implementation. For process querying, experiments show that the technique in this thesis performs on average two orders of magnitude faster than current techniques [10]. For process similarity search, experiments show that the technique in this thesis performs on average one order of magnitude faster than current techniques [30]. The experiments show that the proposed techniques enhance the efficiency of process retrieval.

The third contribution of this thesis is the enhancement of the BP Model
Chapter 9. Conclusions

Repository technology. We have integrated a process-specific index, PFIndex, into APROMORE, an advanced process model repository [53], which is the only BP Model Repository (among the BP Model Repositories we have studied) providing a process-specific index after integration. Therefore, we conclude that the techniques presents in this thesis enhance the BP Model Repository technology in the aspect efficient process retrieval.

9.3 Limitations

As explained in the previous section, the main contributions of this thesis are a framework and survey of BP Model Repositories and two efficient process retrieval techniques. This section lists four limitations of these proposed techniques. The first limitation is about the framework and survey and the last three are about process retrieval techniques.

The first limitation is that the survey of BP Model Repositories in this thesis focuses on existing academic prototypes. However, as business process modeling is becoming an important paradigm for describing and organizing enterprises, BP Model Repository technology becomes more and more important in practice too. Therefore, the industrial tools should also be included in future work. In practice, BP Model Repositories are used as repositories for process modeling tools (such as Aris) or as repositories for process modeling and execution tools (such as IBM WebSphere). We expect that including industrial tools would not influence the framework of BP Model Repositories we have proposed because process modeling and execution tools are already in the framework. We also expect that industrial tools would not provide process specific indexing techniques for process retrieval.

The second limitation is that label matching in this thesis is measured based on syntax, which is difficult to match labels with the similar meaning but written in different terminology, e.g., using synonyms. However, different collections of process models may use different terminology.

The third limitation is that the process retrieval techniques in this thesis only consider activity and control flow aspects. However, process models often contain information about more aspects that may be exploited for process retrieval (nine more aspects are listed in Chapter 2, e.g., resources and data).

The fourth limitation is that the feature-based index, PFIndex, only enhances
the efficiency of process retrieval focusing on process definitions. However process retrieval can also focus on process instances and traces.

9.4 Research Evaluation

To enhance the BP Model Repository technology, the approach described in Chapter 1 is used. This approach at a high level consists of three steps: a state-of-the-art analysis of current BPM Model Repository technology; a proposal for improving this technology; and a check of whether the proposal indeed leads to an improvement.

A methodological weakness with respect to the state-of-the-art analysis is that it does not include functionality that is under research but not integrated by BP Model Repositories yet. This is due to the search strategy that was used when performing the state-of-the-art analysis, which prescribed the use of words related to 'repository'.

A methodological weakness with respect to the evaluation of the proposed improvements is that people developing the process querying technique also were also involved designing the experiments, which could make the experiments biased. To lower the bias, the queries in the experiment are adapted from [6] and [10] and the collection of process models are from the collection of SAP reference models. Another weakness with respect to the evaluation is that the parameters that influence the quality of the results of process similarity search are tuned and optimized during the evaluation. Therefore, the parameters may not be optimized for other experiments in terms of result quality. However, this does not affects the efficiency of the process similarity technique, because the execution of algorithms for the process similarity technique does not change when the collection of process models changes.

9.5 Future Work

Taking into account the limitations and research evaluation, the following can be identified as future work.

Firstly, in this thesis the survey only includes academic prototypes, which should be extended by including existing industrial BP Model Repositories. As such, the
common points and differences between academic and industrial BP Model Repositories can be identified. The framework of BP Model Repositories should also be extended by including functionality of industrial BP Model Repositories.

Secondly, in this thesis label matching is performed by comparing strings. This form of matching can be improved by comparing the labels on a more semantic level, for example by considering synonyms and domain ontologies, which can be done by replacing the node matching functions in this thesis with functions proposed in [31, 34, 37].

Thirdly, process retrieval should consider more aspects besides the activity and control flow aspects. A possible direction is to use an advanced version of the process graph that can record all aspects under consideration, as defined in [31]. Then, features, subgraphs of a process graph, can also record these aspects. After extending the definition of feature matching, features with more aspects can be retrieved through a PFIndex.

In addition, Chapter 2 identifies a number of useful topics in the domain based on literature study. This thesis only consider one of them, process retrieval. More topics about BP Model Repository technology can be researched, as listed in the framework. Furthermore, new functions not in the framework have been proposed, such as, automated process conformance checks [76], process mining [2], process difference analysis [29] and process refactoring [32, 92].
Bibliography


Appendix A

Details of Survey

A.1 Strategy for Literature Search

The literature search strategy that we used is based on [51]. We have searched for information about existing BP Model Repositories, using combinations of the terms ‘business process’, ‘process’, ‘business process model’ and ‘repository’, ‘library’ or ‘reuse’. The search was performed with both Google Web and Google Scholar and the first thirty results were considered for each combination of terms.

A document is considered to be relevant if and only if it satisfies the following four criteria.

1. The project described in the document implements a BP Model Repository or at least proposes a framework or architecture for a BP Model Repository (a BP Model Repository contains a collection of business process models and provides functionality for managing these models).

2. The project described in the documentation is an academic prototype and is presented in refereed publications. We focus on academic prototypes rather than industrial strength repositories, because we assume that academic prototypes present the largest collection of repository functions, considering that it takes time to develop and market new functionality in industry strength tools.

3. The project described in the document should have related refereed publications to ensure its quality.
4. The project documentation is written in English.

For the documents that are selected in this manner, we also include related work and work that cites the document according to Google Scholar and that satisfies the above four conditions.

Table A.1 presents the details of how we have performed the literature search and what the results were. The columns show the literature found, the search terms and strategy that we used, the resources we used to search, and the criteria we use to include and exclude literature. Finally, for each project described presented in the table, we searched for the project name in both Google Web and Scholar for more documents about the project (Table A.1 does not show the documents found in this step).

<table>
<thead>
<tr>
<th>BP Model Repository</th>
<th>Searching Terms/Strategy</th>
<th>Resources</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBPR [60]</td>
<td>‘Business Process Repository’</td>
<td>Google Web</td>
<td>1-4</td>
</tr>
<tr>
<td>BPEL Repository [88]</td>
<td>‘Business Process Repository’</td>
<td>Google Web</td>
<td>1-4</td>
</tr>
<tr>
<td>PVR [55]</td>
<td>‘Business Process Repository’</td>
<td>Google Web</td>
<td>1-4</td>
</tr>
<tr>
<td>APROMORE [53]</td>
<td>‘Business Process Model Repository’</td>
<td>Google Web</td>
<td>1-4</td>
</tr>
<tr>
<td>Fragmento [80]</td>
<td>‘Process Library’</td>
<td>Google Web</td>
<td>1-4</td>
</tr>
<tr>
<td>Reuse Architecture [61]</td>
<td>‘Process Reuse’</td>
<td>Google Web</td>
<td>1-4</td>
</tr>
<tr>
<td>Querying Framework [69]</td>
<td>‘Process Reuse’</td>
<td>Google Web</td>
<td>1-4</td>
</tr>
<tr>
<td>Prosero [38]</td>
<td>Cites [60]</td>
<td>Google Scholar</td>
<td>1-4</td>
</tr>
<tr>
<td>Oryx [8]</td>
<td>Cites [60]</td>
<td>Google Scholar</td>
<td>1-4</td>
</tr>
<tr>
<td>BPCM [43]</td>
<td>Cites [60]</td>
<td>Google Scholar</td>
<td>1-4</td>
</tr>
<tr>
<td>RepoX [83]</td>
<td>Related to [60]</td>
<td>Google Scholar</td>
<td>1-4</td>
</tr>
<tr>
<td>OSIRIS [91]</td>
<td>Related to [60]</td>
<td>Google Scholar</td>
<td>1-4</td>
</tr>
<tr>
<td>Workflow repository [56]</td>
<td>Related to [60]</td>
<td>Google Scholar</td>
<td>1-4</td>
</tr>
<tr>
<td>ProcessGene [90]</td>
<td>Related to [88]</td>
<td>Google Scholar</td>
<td>1-4</td>
</tr>
<tr>
<td>BPMN Repository [85]</td>
<td>Related to [53]</td>
<td>Google Scholar</td>
<td>1-4</td>
</tr>
</tbody>
</table>
A.2 Intermediate Framework

Table A.2 and Figure A.1 show the intermediate management model and reference architecture proposed by us after studying literature about general repositories [19, 77].

<table>
<thead>
<tr>
<th>Data model</th>
<th>Function model</th>
<th>Management model</th>
<th>Repository management</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Not specified]</td>
<td>Storage functions</td>
<td>DBMS</td>
<td>View</td>
</tr>
<tr>
<td></td>
<td>Retrieval functions</td>
<td></td>
<td>Integrity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Transactions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Checkin/out</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Version</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Notification</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Context</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dispatch (Workflow)</td>
</tr>
</tbody>
</table>

A.3 Introduction of Existing BP Model Repositories

In this section we briefly introduce the academic prototypes for BP Model Repositories that we studied. We explain each BP Model Repository according to the following structure. First, we provide a brief introduction of the Repository. Then, we describe how it implements the functionality of the BP Model Repository manager model that is introduced in section 2.3.
A.3.1 The Process Handbook

The process handbook is a knowledge base of process descriptions. The focus of the process handbook is on organizing knowledge about processes, not on providing detailed process models. Consequently, the repository is text-based rather than model-based and the meta-model mainly contains classes storing process models and their descriptions for relating process models to each other and to their
elements. Process models are provided to the process handbook through proprietary tools, in which the information entered in a text-based manner. Internally, the process models are stored in a general database.

The process handbook contains an elaborate classification scheme, which is organized on two dimensions: specialization and decomposition. Specialization allows specialized and, conversely, generalized versions of process models to be considered, like procurement can be specialized into procurement of non product-related goods. Decomposition allows an activity to be decomposed into parts, like procurement can be decomposed into drafting requirements, selecting vendors, ... Along these two dimensions, a number of classifications are developed, such as a classification that decomposed operations into business functions and activities, which can subsequently be specialized for a particular industry or company.

Asides from standard functionality for storing and retrieving information about process models, the process handbook supports browsing the process models along the two dimensions. For each process model the user can browse generalizations, specializations, parts and wholes of which the process model is itself a part. In addition to that the process handbook supports text-based search.

A.3.2 The Process Reuse Architecture

The process reuse architecture [41] is an architecture and prototype tool meant for storing reusable process models, either as a whole or in part.

The main type of artifact that the process reuse architecture considers is the configurable reference process (which it calls process framework). To configure a reference process model, the process reuse architecture also stores process patterns and company specific process models. These can be reused either as a whole or in part to adapt a configurable reference process to the requirements of a particular organization. Like the process handbook, the process reuse architecture focuses on describing rather than modeling processes. However, the process reuse architecture has a more elaborate set of concepts for creating those descriptions, covering description of: process characteristics, organization structure and relation of process models to strategic goals and controls [42]. In addition to concepts for describing process models, the process reuse architecture uses reuse guidelines to describe configuration points in the reference architecture. The different types of process models can be stored in the database using an XML-based file format.
The process reuse architecture supports standard functionality for storing and retrieving process information. In addition to that it allows text-based search and navigation of the repository via three classifications. Process models and their elements are classified by means of the actions that they can perform and the objects on which they perform them. Actions are stored along with synonyms, allowing for more user-friendly search. Process patterns are classified as individual (regular) patterns, pattern families, which represent a sets of patterns that can be used in combination to solve a larger problem and community patterns, which are patterns targeted towards a particular problem domain.

A.3.3 The Library for Process Programming

The library for process programming \[103, 104\] is an architecture for organizing business process models into libraries from which individual business process models can be reused. The system provides four mechanisms for reusing existing process definitions: inheritance, nesting, integration, and reflection.

The library for process programming focuses on the definition of processes, the enactment of process instances and adaptation of processes for reuse. For the purpose of defining processes, the system introduces a language called ‘P’, which resembles a textual programming language. It focuses on the definition of processes and activities, control flow and data used by processes.

Inheritance and nesting of processes lead to tree structures that can be traversed. Such tree structures can be used to reveal the configuration of a process as it is composed of other processes and activities.

A.3.4 The Repository for Integrated Process Management

The repository for integrated process management (IPM) \[24, 26\] is a repository for managing business process models throughout their lifecycle. As such it does not only have basic functions for storing and retrieving business process models, but also advanced functions for version and configuration management.

Process information can be exchanged with the repository through the IPM Executable Process Definition Language (IPM-EPDL) \[24\], which is an XML-based format which can be used to store information about process models and activities, control flow, organizational structure, authorization and resource assignment,
data and monitoring. Internally, the repository has connectors for storing process information in an XML database or a relational database. IPM focuses on storing company specific process models and running instances of those processes.

The IPM repository has extensive support for storing and retrieving process models and related information. For that purpose a query language, the IPM Process Query Language (IPM-PQL), has been defined, which has support for process specific queries like searching processes that have a certain activity or a certain transition from one activity to another. IPM also supports browsing processes in the repository using a number of classifications.

The IPM repository has two interfaces, an external interface that uses the IPM-EPDL to exchange process information with external tools such as modelling, analysis and enactment tools and an internal interface to connect the repository with a database.

The IPM repository has support for three management functions: lifecycle, versioning and configuration management. The lifecycle management function ensures that processes go through a number of states during their life, such as 'in design', 'validated' and 'in operation'. It also ensures that certain operations can only be performed on a process in certain states. For example, only validated processes can be enacted. Version management allows multiple versions of a process to be created and configuration management enables users to maintain relations between (versions of) processes and (versions of) the sub processes of which they are composed.

### A.3.5 RepoX

RepoX [83] is an XML-based process model repository, which is a part of the METEOR workflow system. Although it is part of the METEOR project, it has been specifically developed with the intention to standardize the exchange of process models between a process definition tool and a workflow engine (known as interface 1 of the workflow reference model [94]).

Process models can be exchanged with the repository, using a XML documents in a pre-defined format. Internally, the models are stored in an object-oriented database. RepoX stores the control flow aspect of process models along with the data that is used in the processes and the roles that are authorized to perform tasks in the processes. Since RepoX is primarily meant as the repository for a workflow
engine, it stores company-specific process models.

RepoX has functionality for version and configuration management. Versions of individual tasks, sub-processes, and full processes can be stored in the repository, along with configuration information about which version of a task or sub-process is a part of which version of a process. RepoX also supports check-in and check-out of processes and their elements.

A.3.6 The Repository for Workflow Systems

In [56] a repository for workflow systems is described. The project provides a conceptual model for storing workflow models, requirements for managing workflow model repositories, and an architecture design for implementing a repository manager.

The repository supports the activity, control flow, data, resource and monitoring aspects, and stores workflows in an object-oriented database. As defined in its workflow metamodel, it considers a workflow specification or resource as an entity, a workflow task as a weak entity, and constraints between tasks as relationships between entities. A workflow specification contains several workflow tasks, and constraints can be sequential composition of tasks, conditional choice between tasks, iterative performance of a task, split (and/or) between tasks and join (and/or) of tasks.

The Object-oriented Database Management System (OODBMS) provides the support the basic functionality for storing and retrieving workflows. Also, with help of scheduling, tracking and administration tools, the repository can record and manage monitoring information during the workflow runtime.

To support version and configuration managements, the repository defines a class named ‘versionedObject’ to record the version information, and the class also defines several functions to provide checkin/checkout and notification functions.

A.3.7 The BPMN Repository Architecture

The BPMN repository architecture [85, 86] is a repository architecture, rather than a complete implementation. It is specifically targeted towards BP Model Repositories that support inter-organizational processes, i.e.: processes or compositions of multiple processes that span organizational boundaries.
The BPMN Repository Architecture proposes that a broad range of types of process artifacts and aspects should be supported for that purpose. Process aspects that should be supported are the control flow aspect, the data aspect, the organizational structure and authorization information, monitoring information and controls. Types of artifacts that should be supported are reference models, company specific models, process instances and historical information about process instances. The BPMN Repository Architecture prescribes that processes and related information should be stored in a standardized XML format. In particular it proposes the BPMN, along with a standardized BPMN XML interchange format, as the standardized set of concepts.

The BPMN Repository Architecture proposes four architectural elements that are specific for the support of inter-organizational processes. Firstly, it proposes that ontologies should be stored in the repository. Ontologies should facilitate the integration of processes from multiple parties by semantically matching different terminology of different parties. Secondly, it proposes that a distinction is made between private repository information that is available only to a single party and public repository information that a party makes available to its partners. Thirdly, it proposes that a distributed repository is used, which corresponds to the distributed nature of inter-organizational processes. Fourthly, it proposes that parties that are involved in the processes can specify the processes in their own modeling notation of choice and that transformations are used from those notations into the common BPMN XML format.

A.3.8 Oryx

Oryx [28, 72] is a web-based process modeling tool that supports users browsing, creating, storing and updating process models online. The tool uses a repository for storing the business process models that are created with it.

Oryx mainly focuses on the activity and control flow aspect; it stores company-specific process models. It supports many (process) modeling notations, including: BPMN, EPC, Petri nets, Workflow nets, FMC Block Diagrams and XForms. Internally, processes are stored in a database; externally it represent process models in RDF format.

Oryx does not only provide basic storage functions (create, update and delete), but also implements import and export functions. For example, for the BPMN
notation, Oryx can import processes from ERDF and JSON formats and processes can be transformed from BPEL. It can export process models in ERDF, JSON, RDF, PNML, XPDL, and XHTML formats or convert to Petri nets. For querying business processes, Oryx integrates BPMN-Q \cite{6-9} as part of the project. BPMN-Q is a visual query language for business process models, which extends the language elements of BPMN to support queries.

### A.3.9 BP-Suite

BP-Suite \cite{17} is a tool suite based for execution of processes specified in BPEL. It consists of BP-QL (a query language for business process definitions) \cite{14}, BP-Mon (a tool for monitoring running business process instances) \cite{15-17} and BP-Ex (a tool for analyzing the logs of the executions) \cite{17}.

BP-Suite supports the activity, control flow, data, resource, organizational structure, monitoring and authorization aspects. It supports storing process models, process instances and historical information about process instances. Process models are stored in the BPEL XML format.

Asides from the standard database functionality for storing and retrieving process models, BP-Suite supports queries on all three types of processes that it can store (definitions, running instances and logs).

### A.3.10 ProcessGene

The ProcessGene project \cite{90} provides a tool for querying business process models. It consists of four parts: a Scoping-Assistant (SA), a Query Specification Interface (QSI), a Query Interpreter (QI) and a Query Results Packager (QRP). Users of the ProcessGene provide querying scope and specifications by the SA and QSI; then the QI compiles specification to querying requirements and the QRP returns querying results.

ProcessGene focuses on the activity, control flow and authorization aspects and supports reference and company specific process models.

ProcessGene supports standard database functionality for storing and retrieving process models. In addition to that it is meant for querying business process models.
A.3. Introduction of Existing BP Model Repositories

A.3.11 The Process Variants Repository

The Process Variants Repository (PVR) \[57, 58\] provides mechanisms for dealing with different variants of a business process model at run time. For this purpose, it provides functionality for storing a (company-specific) business process model, along with (restrictions on) allowed variations of that business process model. Historical information about variations that are performed at run-time is also stored, such that it can be used to improve the business process model.

As defined in the process variant schema a variant consists a process model, an execution sequence based on the process model, resources, data, tasks, a design description and authorizations. Accordingly, PVR supports the activity, control flow, data, resource, monitoring and authorization aspects and stores company-specific processes, process instances and historical information about process instances.

Although PVR is designed for querying process variants (logs), it also provides an support for querying process definitions. In addition to that it provides standard functionality for storing and retrieving process models and their variants.

A.3.12 The Querying Framework

The Querying Framework \[65, 66\] is a framework for developing advanced querying mechanisms for a business process repository.

The Querying Framework supports the activity, control flow, data, goal, resource and authorization aspect. The resource aspect includes information resources. Hence, the data aspect is supported at a high level of abstraction. The Querying Framework stores information about these aspects in an RDF-based \[21\] ontology repository. The language that is used to enter the information is the WSML \[22\]. Information can be stored about process patterns, process fragments, company specific processes and, although they are not explicitly mentioned, reference processes.

The repository can be queried using WSML Logical Expressions. The Querying Framework has specific support for searching process fragments that complete a selected (incomplete) process model part and for searching substitutions for a selected process model part.
A.3.13 The Semantic Business Process Repository

The Semantic Business Process Repository (SBPR) [49, 60] is an ontology-based repository for storing business process models.

The SBPR does not commit to a particular set of aspects of business process models that must be stored. Instead, it requires that the repository is configured with a process ontology, of which the concrete process models are be instances. In particular, it considers BPMO [45], sBPMN [71], sEPC [50] and sBPEL [4] as ontologies. These ontologies must be specified in WSML. Internally, business process models are stored in a relational database.

Aside from the standard CUD functionality, the SBPR supports querying, using WSML as a query language, versioning and check-in/check-out management.

A.3.14 The BPEL Repository

The BPEL repository is a process repository, developed as an Eclipse plug-in, which is designed for storing and retrieving business processes in the BPEL format along with associated metadata [87, 88].

The repository uses the BPEL XML format as its external (interchange) format and stores the process models and their elements internally as objects in an EMF repository.

Besides the standard functionality for storing and retrieving process models, the BPEL repository can interact with query engines that are built on the EMF repository (such as the Kent OCL engine), EMF extensions and other external software.

A.3.15 Prosero

The Prosero project [38] combines business process management and Service Oriented Architecture (SOA) technology to support business process outsourcing based on Web Services.

The (semantic) repository of Prosero consists of four components: the terminology, the Reference Model Repository (RMR), the Web Service Repository (WSR) and Customer Model Repository (CMR). These components can be used to store reference process models and company-specific process models and process instances. Prosero has an external interface for modeling tools (organizational
structure modeling, data modeling and business process modeling) and the ActiveBPEL4People engine. The aspects that are supported by Prosero are the activity, control flow, data, resource, authorization and organizational structure aspect. Prosero stores BPMN processes and executes BPEL processes, consequently it has a BPEL generator, which can transform BPMN into BPEL.

Prosero support standard functionality for storing and retrieving process information. With the help of the ActiveBPEL4People engine, Prosero can deploy, automate and analyze processes.

A.3.16 OSIRIS

OSIRIS (Open Service Infrastructure for Reliable and Integrated process Support) \cite{78, 79, 91} has been proposed for peer-to-peer process execution. The process repository support that it provides focuses on storing business process models, service specifications as they are provided or used by business processes and instances of executing business process models. In addition to that, OSIRIS supports typical peer-to-peer functions, such as concurrency control and load balancing.

OSIRIS supports the activity, control flow and data aspect and stores company specific process models (or services) and running process instances. Internally, process models and service specifications are stored in a database.

In addition to standard functionality for storing and retrieving process models, OSIRIS has an interface with a modeling tool, named O’Grape, which uses a proprietary notation.

A.3.17 APROMORE

APROMORE (Advanced PROcess MOdel REpository) \cite{53} has been proposed for supporting multiple process modeling notations and providing a scalable architecture that integrates management functionalities for process models. To achieve these goals, APROMORE provides a canonical process format, which can represent most elements of six business process modeling notations, including EPC, BPMN 1.2, Protos 8.0, WF-Nets, YAWL 2.0, and WS-BPEL 2.0. In addition, APROMORE is implemented based on a service oriented architecture.

APROMORE supports the activity, control flow, data and resource aspect and stores company specific process models, reference models and process patterns.
Appendix A. Details of Survey

Process models are stored in XML files externally and in a relational database internally.

APROMORE supports importing and exporting process models, besides standard functionality for storing process models. APROMORE also provides categories based on the business semantics of process models for retrieving process models.

A.3.18 Fragmento

Fragmento is a prototype for a process library, which manages characteristic process fragments. The fragments in Fragmento have the following characteristics. The fragments, with or without constrained regions, are approved according to their particular situations; the fragments observe the 4-eyes principle; the fragments store time-stamps; the fragments avoid infinite waits. The purpose of Fragmento is to reuse process fragments in the library.

Fragmento supports the activity, control flow, data, resource, and authorization aspect; stores company specific process models and process patterns in XML (BPEL) externally and object-relational database (PostgreSQL) internally.

Fragmento supports all of five functionalities for storing process models in the framework. It also supports process version management.

A.3.19 BPCM

The BPCM (Business Process Characterizing Model) repository provides an architecture for managing BPCMs, as such BPCMs can be reused.

A BPCM meta model has been proposed, which supports five process aspects, i.e., activity, resource, authorization, goal, semantic. The BPCM repository stores company specific process models in XML externally and relational database internally.

The BPCM repository supports importing and exporting process models, besides standard functionality for storing process models. It also supports process version, configuration, and view management functionalities.

A.3.20 IPR

IPR is an Integrated Process Repository of multiple process reference models, e.g., MIT process handbook and SAP reference models. The purpose of IPR is to
support process design for e-business by reusing reference models. IPR provides a comprehensive classification mechanism for reference models, which makes it easier to find related reference models with respect to the desired one.

IPR supports all the aspects that process reference models support, including activity, control flow, data, resource, and relationship. IPR stores process reference models as well as the company-specific models after adapting reference models. IPR stores process models in XML externally and repository internally.

IPR supports standard functionality for storing process models. It provides a classification mechanism for quickly searching and navigating process models.
Summary

As organizations increasingly work in process-oriented manner, the number of business process models that they develop and have to maintain increases. As a consequence, it has become common for organizations to have collections of hundreds or even thousands of business process models. When a collection contains such a large number of business process models, it is impossible to manage that collection manually. Therefore, Business Process (BP) Model Repositories are required that store large collections of process models and provide techniques for managing these collections automatically and efficiently.

The goal of research described in this thesis is to improve on existing BP Model Repositories, by improving the management techniques that are supported by these repositories on an aspect that has received little attention so far. Looking ahead at the results of the research, the aspect that will be selected for improvement is the process retrieval aspect.

The two main research activities that will be carried in the context of this research are the following.

Firstly, a survey of Business Process Model Repositories is performed to identity an unsolved aspect to be enhanced. The functionality of existing BP Model Repositories is listed and summarized as a framework for BP Model Repositories. After comparing the functionality that is provided by existing BP Model Repositories, based on the framework, efficient process retrieval is selected as the aspect that will be improved. This aspect is selected, because, although existing BP Model Repositories provide techniques for process retrieval, none of them focus on the efficiency of process retrieval.

Secondly, an indexing technique for process retrieval (both process similarity search and process querying) is proposed. The index is constructed using features
of process models. Features are small and characteristic fragments of process models. As such, by matching features of a given query/search model and features of a process model in a collection, a small set of models in the collection that potentially match the query/search model can be retrieved efficiently through the index. Techniques are also proposed to further check whether a potential match is an actual match for the query/search model. All of the above techniques are implemented as a component of the APROMORE (an Advanced Process Model Repository) process repository.

To evaluate the proposed process retrieval techniques, experiments are run using both real-life and synthetic process model collections. Experimental results show that on average the process retrieval techniques proposed in this thesis performs at least one order of magnitude faster than existing techniques.
Acknowledgements

It is a great pleasure for me to do my PhD project at the Eindhoven University of Technology in the Netherlands. The thesis you are holding is the result of my research for the last four years, which would not exist without the help of many people. I would like to thank all these people who have supported me during my PhD study.

First of all, I would like to thank my promoter, Paul Grefen, who accepted me as a PhD in the IS group four years ago. I have learned a lot from Paul, both research and culture. He taught me the way a PhD should think and explained me the differences between Dutch and Chinese cultures. I would like to thank my supervisor, Remco Dijkman, who drove me to my apartment from the train station of Eindhoven the first day I arrived and have driven me all the way to obtain a PhD in the last four years.

Further, I would like to thank Mathias Weske, Manfred Reichert, Uazy Kaymak, Wil van der Aalst, and Arjen de Vries for being in my PhD committee; thank Will Bertrand for chairing my PhD defense.

Also, I would like to thank my colleagues in the Information Systems group. Joel Ribeiro, Heidi Romero, Jana Samalikova, Egon Luftenegger, Ronny Mans, Ricardo Seguel Perez, Rob Vanwersch, Nan Shan, Vassil Stoitsev, Macro Comuzzi, Irene Vanderfeesten, Pieter van Gorp, Hajo Reijers, Arthur ter Hofstede, Annemarie van der Aa, and Ada Rijnberg-Wielaad, thank you all for having lunches and sharing wonderful memories with me these years.

Then, I would like to thank my close friends in Eindhoven, who have dinners and play with me every weekend. Zhao Kang, Wang Yi, Yang Dujuan, Zhang Rui, Tian Tian, Peng You, Yang Yang, Li Shiqi, Kong Chuipeng, Zhang Mingshuo, Liao Feixiong, Yu Chen, Hao Chenlei, Li Jing, Hui Jianhua, Tong wei, and Gao Hao,
thank you all for the companion all these years. I would like to thank my roommate Liu Zheng for sharing an apartment with me. I would like to thank my friends in the Association of Chinese Students and Scholars in Eindhoven (ACSSE). Li Jing, Xu Wei, Man Jinfan, Zhang Peng, Li Yan, Lin Yuzhong, Zhang Ying, Wang Jia, Wang Jingru, Zheng Qiang, Sui Huapeng, and Liu Caixia, it has been a great time to work with you. I would like to thank everyone in the football and badminton teams for the games we have played together.

Finally, my special thanks go to my grandmother, my parents, and my relatives, who have always been supporting me unconditionally.

谨以此书献给一直以来疼爱我的奶奶、爸爸、妈妈和亲属们，感谢你们二十多年来对我的支持和照顾！

Yan Zhiqiang
July 2012
Curriculum Vitae

Zhiqiang Yan was born on 29 November, 1983 in Panjin, Liaoning Province, People’s Republic of China. After finishing high school education at Panjin High School in 2002, he studied Computer Science in an honor class at Northeastern University in China, where he received his Bachelor and Master degrees in 2006 and 2008 respectively. In 2008 he won a scholarship from China Scholarship Counsel (CSC) to start a PhD project at Eindhoven University of Technology in the Netherlands of which the results are presented in this dissertation.