FNet: an index for advanced business process querying

Yan, Z.; Dijkman, R.M.; Grefen, P.W.P.J.

Published: 01/01/2012

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the author’s version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 16. Feb. 2017
FNet: An Index for Advanced Business Process Querying

Zhiqiang Yan, Remco Dijkman, Paul Grefen

Beta Working Paper series 385

<table>
<thead>
<tr>
<th>BETA publicatie</th>
<th>WP 385 (working paper)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISBN</td>
<td></td>
</tr>
<tr>
<td>ISSN</td>
<td></td>
</tr>
<tr>
<td>NUR</td>
<td>982</td>
</tr>
<tr>
<td>Eindhoven</td>
<td>June 2012</td>
</tr>
</tbody>
</table>
FNet: An Index for Advanced Business Process Querying

Zhiqiang Yan, Remco Dijkman, and Paul Grefen
Technology University of Eindhoven
PO Box 513, NL-5600 MB Eindhoven, The Netherlands
z.yan,r.m.dijkman,p.w.p.j.grefen@tue.nl

Abstract. Nowadays, more and more organizations describe their operations in terms of business processes. Consequently, it is common for organizations to have collections of hundreds or even thousands of business process models. This calls for techniques to quickly retrieve business process models that satisfy a given query. Some advanced techniques for querying a collection of business process models exist. However, these techniques focus on the expressive power of the query language, rather than the efficiency of retrieval of models that satisfy the query. Consequently, querying a collection of models can take considerable time. To solve this problem, this paper proposes an efficient technique, using feature nets. Experiments show that on average the technique performs two orders of magnitude faster than existing techniques.

1 Introduction

Nowadays, business process management becomes more and more important in managing organizations. To increase the flexibility and controllability of the management of organizations, business processes are used to describe their operations. As a result, it is common to see collections of hundreds or even thousands of business process models. For example, the collection of SAP reference models consists of more than 600 business process models [14], and the collection of the reference models for Dutch Local Government contains a similar number of models [8]. As business process model collections increase in size, business process model repositories are developed to provide process-specific functions managing them. Querying a collection of business process models is one of these functions [22].

Querying a collection of business process models is done by providing a business process model fragment as a query. The querying technique then returns all business process models from the collection that contains that fragment. For example, if query $a$ from Fig. 1 is provided to the querying technique, then the first three models should be returned, because they all contain that fragment; graph 4 should be returned for query $b$.

This paper focuses on advanced business processes queries, which are queries that contain advanced query modeling elements, e.g., query $b$ from Fig. 1. There are four advanced query elements, which will be defined in section 3: wildcard
nodes, transitive edges, negative edges, and negative transitive edges. To the best of our knowledge, three advanced query languages exist [1,3,6]. However, performing an advanced business process query using these techniques 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 [2,15]. While users of a search engine typically expect a response within milliseconds.

Therefore, our goal is to make advanced business process querying more efficient. To this end, this paper introduces the concept of feature net (FNet for short) and an efficient technique for querying based on feature nets. In this way, while both advanced query languages and techniques for efficiently performing non-advanced queries, the contribution of this paper is a technique for efficiently performing advanced queries.

The rest of the paper is organized as follows. Section 2 introduces the concept of process graph, which we use as the underlying formalism to define our efficient querying technique. Section 3 introduces query process graphs. Section 4 introduces the concept of feature of a process (query) graph. Features are the elements of an FNet. Section 5 presents the FNet and shows how it can be constructed for a collection of process models and how it can be used to make advanced querying more efficient. Section 6 presents the implementation of the FNet. Section 7 presents the synthetic process model generator, which can generate process models for evaluating the FNet. Section 8 evaluates the performance of the FNet. Section 9 introduces related work and Section 10 concludes the paper.

2 Process Graph

We define our querying technique on process graphs. 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 the structure of existing (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, Fig. 9 shows the business process graphs for the models from Fig. 1. As shown
in graph 4 of Fig. 9, we assign each gateway node a unique label to represent its routing function, e.g., ‘And-Split’ and ‘Xor-Join’.

**Definition 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 $n\bullet = \{m | (n,m) \in E\}$ is the post-set of $n$.

![Fig. 2. Querying a Collection of Business Process Graphs](image)

A path is a sequence of edges. For example, in graph 4 of Fig. 9, there is a path from node ‘Order Goods’ to node ‘Receive Goods’.

**Definition 2 (Path).** Let $G = (N,E,\lambda)$ be a process graph and $n_1,n_s \in N$ be nodes of $G$. There is a path from $n_1$ to $n_s$ if and only if there exists nodes $\{n_1,n_2,n_3,\ldots,n_s\} \subseteq N$ ($s > 1$), and $\{(n_1,n_2),\ldots,(n_{s-1},n_s)\} \subseteq E$.

### 3 Query Process Graph

Being a fragment of a business process, a business process query 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 [1,3,6] also use the following types of nodes and edges: wildcard node, transitive edge, negative edge, and neg-transitive edge. A **wildcard node** matches any node in a business process graph. A **transitive edge** matches a path 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 path from its source node to its target node.

**Definition 3 (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.

For example, as shown in Fig. 3, query $c$ has a wildcard node, which is denoted as a node without any label; query $d$ has a transitive edge, $\theta(\text{Order, Pay}) = \text{transitive}$, which is represented as an edge with ‘$*$’ as its label; query $e$ has a negative edge, $\theta(\text{Receive, Pay}) = \text{negative}$, which is represented as an edge with ‘$\neg$’ as its label; query $f$ has a neg-transitive edge, $\theta(\text{Receive, Pay}) = \text{neg-transitive}$, which is represented as an edge with both ‘$\neg$’ and ‘$*$’ as its label. Besides these queries with advanced elements, a query process graph can also be a basic query like query $a$ in Fig. 9.

![Query Process Graphs with Advanced Nodes or Edges](image)

Fig. 3. Query Process Graphs with Advanced Nodes or Edges

Using the definition of a business process graph and a query graph, querying is done by finding business process graphs that match a given query graph.

**Definition 4 (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 $(n_Q, n_G) \in M$, either $\Theta(n_Q) = \text{wildcard}$ or $\omega(\lambda_Q(n_Q)) \subseteq \omega(\lambda_G(n_G))$, where $\omega(l)$ denotes the set of words that appear in a label $l$;
- 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 path from $M(n_Q)$ to $M(m_Q)$ 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 path from $M(n_Q)$ to $M(m_Q)$ in $G$.

\[1\] Label matching can be measured in a number of different ways [7]. For illustration purposes, we perform label matching by considering words of a label. This also allows users to query with only words under their concerns (like Google). It can be easily replaced by other metrics for label matching.
For example, in Figure 9 and 3, node ‘Order’, is matching with nodes, ‘Order Goods’ and ‘Order Goods Online’; ‘Receive’ is matching with ‘Receive Goods’ and ‘Receive Application’; a wildcard node, ‘∗’, is matching with all nodes; ‘Pay’, is matching with ‘Pay’; query a, ‘Order’→ ‘Receive’, is matching with graph 1,2,3; query b, ‘Order’→ ‘Receive’→ ‘Pay’, query c, ‘∗’→ ‘Receive’, is matching with graph 1,2,3,4; query d, ‘Order’→ ‘Pay’, is matching with graph 3,4; query e, ‘Receive’→ ‘Pay’, and query f, ‘Receive’→ ‘Pay’, are matching with graph 4.

4 Features

To efficiently query a collection of business process graphs, we break both the process graphs and the query process graphs up into features. Features should be small and representative. Since they are small, they can be used for efficient processing. Since they are representative, results of a query feature are candidates of results of a query graph. After also defining an index on features in section 5, we can use them for fast query processing. This section presents how to perform feature-based querying.

Taking the criteria for selecting features (small and representative) into account, we only consider features based on the most common workflow patterns: sequence, split, join, and loop. Besides that we also consider single nodes as a feature, because we want to construct an index based on node labels. We name these features basic features.

Definition 5 (Basic Feature). Let \( \mathcal{D} \) be a collection of process graphs and \( g \in \mathcal{D} \) 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 \( s \geq 1 \) 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 max be a threshold of the size of a feature, $\Gamma(g, \text{max}) = \{f \in \Gamma(g)|\text{Size}(f) \leq \text{max}\}$.

For example, for graph 4 in Fig. 9, the set of basic node features consists of nodes ‘Order Goods’, ‘Receive Goods’, ‘And-Split’, and ‘Pay’; the set of the basic sequence features of size 1 consists of sequences ‘Order Goods’→‘And-Split’, ‘And-Split’→‘Receive Goods’, ‘And-Split’→‘Pay’, and ‘Receive Goods’→‘Order Goods’; the basic split feature set consists of the feature with split node ‘And-Split’ and the set of nodes ‘Receive Goods’, ‘Pay’; and the basic loop feature set consists of the loop feature with three basic nodes ‘Order Goods’, ‘And-Split’, and ‘Receive Goods’.

More features are used other than sequences in this paper, which helps filter more graphs that are not matching with a given query [20]. For example, given a split query $a \rightarrow \{b, b\}$, a sequence $a \rightarrow b$ is filtered using split features.

A process graph contains only basic features. However, as explained in Section 3, a query process graph can also contain wildcard nodes, transitive edges, negative edges, and neg-transitive edges. Consequently, we need advanced features to be able to break up a query process graph into query features.

**Definition 6 (Advanced 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 $N_F = \{n\}$ 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.

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 $qg$.

The basic features in Definition 6 also includes basic features with wildcard nodes besides features in Definition 5, which can be used as a query feature. For example, in Fig. 3, query $c$ is a basic query feature; query $d$ is a transitive query feature; query $e$ is a negative query feature; query $f$ is a neg-transitive
query feature. Definition 4 is used to measure whether an advanced feature is matching with a basic feature.

In order to optimally benefit from indexes that can be constructed for features, we allow features to be constructed hierarchically, such that we can use a multi-level (hierarchical) index. To this end, Definition 7 defines a hierarchical relation between features.

**Definition 7 (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.

For example, for graph 4 in Figure 9, the direct child feature set of the node feature ‘Receive Goods’ is the set consisting of sequence features ‘And-Split’→‘Receive Goods’ and ‘Receive Goods’→‘Order Goods’.

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 4.

More precisely, feature-based querying is defined as follows.

**Definition 8 (Feature-based Querying).** Given a business process graph $g$, a query graph $qg$ 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 $\{af_1, af_2, \ldots, af_n\}$ (as defined in Definition 6). 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 4;
- 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 $f_i$, then $M(n) = M_i(n)$ ($1 \leq i \leq n$);
- for the mapping $M$, each advanced query feature $af_i$ is matching with the process graph $g$ according to Definition 4.

For example, Figure 4 shows a query process graph and three process graphs. If we use only nodes and sequences of size 1 as features, then query $g$ 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, $b \not\rightarrow d$. First, 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. Second, 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. Third, 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.

---

Fig. 4. Example of feature-based querying

---

5 Feature Net

In order to speed up the querying operation, we must be able to quickly determine which process graphs in a collection contain all the features of a query graph without contradictions, because, according to Definition 8, those process graphs are the results for the query. In order to determine this, we construct an index of all basic features and transitive sequence features that process graphs in the collection contain. We call this index the feature net (FNet).

Figure 5 shows how this works. There are two operations that can be performed on the FNet: indexing and querying. When indexing a collection of process graphs \(\{G_1, G_2, \ldots\}\), an FNet is constructed that consists of the process features, \(PF_1, PF_2, \ldots\), and a mapping to the graphs that have those features. When querying, the query graph must be split up into query features, \(QF_1, QF_2, \ldots\). For each of those features, the matching process features (if any) are then determined. Subsequently, those process graphs are returned that are a match for all of the features.

In this section we describe in more detail how an index of features, an FNet, can be constructed and how that FNet can be queried.
5.1 Constructing an FNet

An FNet 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 FNet maps each feature to the process graphs of which it is a feature and each node of a feature to the process graph node that it represents. This mapping is required to decide which feature nodes also represent the same graph node, which must be checkable according to the second requirement of definition 8.

More precisely, an FNet is defined as follows.

Definition 9 (Feature Net (FNet)). Let $\mathcal{D}$ be a collection of process graphs with disjoint sets of nodes. The feature net of $\mathcal{D}$, denoted $\text{FNet}(\mathcal{D})$, is a tuple $(F, RF, \upsilon)$, in which:

- $F = \bigcup_{G \in \mathcal{D}} \Gamma(G)$ is the set of all basic features of graphs in $\mathcal{D}$, where $\Gamma(G)$ returns the basic feature set of $G$.
- $RF = \{(f_1, f_2) | f_1, f_2 \in F \land f_1 \in \text{DPFS}(f_2)\}$ is the direct parent-child relation between features as defined in Definition 7.
- $\upsilon : (\bigcup_{f \in F} N_f) \rightarrow \mathcal{P}(\bigcup_{G \in \mathcal{D}} N_G)$ is the function that maps each feature node to the graph nodes from which the feature is derived.

A drawback of an FNet 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 [12]. The inverted index does not contain stop-words, e.g., ‘a’, ‘an’, ‘the’, ‘one’, ... and uses lower case versions of the words.

Algorithm 1 - 4 present the algorithm to construct and manage an FNet. Constructing an FNet consists of the following two steps as shown in Algorithm 1. Firstly, an empty index is initialized. Then, each process graph in the collection is inserted into the index.

Inserting a process graph into an FNet 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 FNet. 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
Algorithm 1: Construct an FNet

\begin{algorithm}
\caption{Construct an FNet}
\begin{algorithmic}[1]
\STATE \textbf{input} : a collection of process graphs: $D$
\STATE \textbf{output} : an FNet: $\text{FNet} = (F, RF, \upsilon)$
\STATE \begin{algorithmic}
\STATE \hspace{1em} $\text{FNet} \leftarrow \text{null}$;
\STATE \hspace{1em} \textbf{foreach} $g \in D$ \textbf{do}
\STATE \hspace{2em} \text{insertGraph} $(g, \text{FNet})$; \text{ // Algorithm 2}
\STATE \hspace{1em} \text{return} $\text{FNet}$;
\end{algorithmic}
\end{algorithmic}
\end{algorithm}

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 ($\upsilon$).

Algorithm 2: Insert a process graph into an FNet

\begin{algorithm}
\caption{Insert a process graph into an FNet}
\begin{algorithmic}[1]
\STATE \textbf{function} \text{insertFeature} $(\mathcal{F}, \text{FNet})$
\STATE \begin{algorithmic}
\STATE \hspace{1em} \textbf{while} $\mathcal{F} \neq \emptyset$ \textbf{do}
\STATE \hspace{2em} \textbf{foreach} $f \in \mathcal{F}$ \textbf{do}
\STATE \hspace{3em} \textbf{if} $\bar{f}_1 \in \mathcal{F}, f_1 \in \text{DPFS}(f) \text{ //Definition 7}$ \textbf{then}
\STATE \hspace{4em} \textbf{if} $f \notin F$ \textbf{then}
\STATE \hspace{5em} $F \leftarrow F \cup \{f\}$;
\STATE \hspace{5em} \textbf{foreach} $f_1 \in \text{DPFS}(f) \subseteq F \text{ //Definition 7}$ \textbf{do}
\STATE \hspace{6em} $RF \leftarrow RF \cup \{(f_1, f)\}$;
\STATE \hspace{6em} $\upsilon(N_f) \leftarrow \upsilon(N_f) \cup \{N_f\}$;
\STATE \hspace{5em} $F \leftarrow F - \{f\}$;
\end{algorithmic}
\end{algorithmic}
\end{algorithm}

Deleting a process graph from an FNet 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 an FNet are given latter in Section 5.2, when presenting feature matching through an FNet). Similarity to 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.

Algorithm 3: Delete a process graph from an FNet

```plaintext
function deleteFeature ($F$, FNet)
begin
while $F \neq \emptyset$ do
    foreach $f \in F$ do
        if $\not\exists f_1 \in F, f_1 \in DCFS(f)$ //Definition 7 then
            if $v(f) = \{N_f\}$ then
                $F \leftarrow F - \{f\}$;
                foreach $f_1 \in DCFS(f) \cap F$ //Definition 7 do
                    $RF \leftarrow RF - \{(f_1, f)\}$;
            else
                $v(N_f) \leftarrow v(N_f) - \{N_f\}$;
        $F \leftarrow F - \{f\}$;
end
end
```

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

begin
$F = \Gamma(g, max)$; //Definition 5.
deleteFeature ($F$, FNet);
return FNet;
```

As described in Algorithm 4, updating a process graph into an FNet 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.

Algorithm 4: Update a process graph in an FNet

```plaintext
input: the original process graph: $g$, the updated process graph: $g_1$, an FNet: $FNet = (F, RF, v)$, a threshold: max
output: an FNet: $FNet = (F, RF, v)$
begin
$F = \Gamma(g, max)$; $F_1 = \Gamma(g_1, max)$; //Definition 5.
deleteFeature ($F - F_1$, FNet); //a function in Algorithm 3
insertFeature ($F_1 - F$, FNet); //a function in Algorithm 2
return FNet;
```

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 an FNet. The number of features 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) an FNet are $O(n + \sum_{1 \leq i \leq max} C_{m}^{max}) = O(n + m^{max})$; both time and space complexities for constructing an FNet are $O(k \cdot (n + m^{max}))$.

The threshold $max$ is a small number, e.g., from 0 to 3 in our experiments in Section 8.

Figure 6 shows an example of a FNet. The FNet is generated based on graph 1-5 in Fig. 9. From left to right, five columns are listed, in which the first one lists the words that appear in labels and the latter four list features ($F$) ordered by their sizes in the ascending order. Between columns, relationships (arrows) are drawn to connect a word to a node feature (the inverted index) and connect a feature to its child features ($RF$). Each feature is associated with both graphs containing the feature and lists of nodes of which the feature consists ($\nu$). For the sake of simplicity, not all of the features are shown in the figure.

5.2 Querying an FNet

To query a collection of process graphs for a given query graph through a FNet, four steps are performed: retrieving nodes, generating features, retrieving features, and checking graphs.

Firstly, for each query node, the words in its label are looked up, using the inverted index, to retrieve matching node features. A node feature is returned if all words in the query node’s label return that node feature. If the query node is a wildcard node, then all the node features in the FNet are returned. The querying proceeds if and only if all the query nodes have matching node features in the FNet; otherwise there is no graph from the collection that satisfies the query. Taking query $a$ and query $b$ in Fig. 9 and the FNet in Figure 6 as an example, query node ‘Order’ has two matching features, ‘order goods’ and ‘order goods online’; query node ‘Receive’ has one matching feature, ‘receive goods’; query node ‘Pay’ has one matching feature, ‘pay’.

Secondly, features are generated from the query graph, by breaking up the query graph into subgraphs that are advanced features, as defined in Definition 6. In the example, query $a$ contains a basic sequence feature, ‘Order $\rightarrow$ Receive’; query $b$ contains a transitive feature, ‘Order $\rightarrow$ Receive’; and a negative feature, ‘Receive $\rightarrow$ Pay’.

Thirdly, the FNet is used to retrieve matching features for each basic query graph feature, as defined in Definition 4. Given a query feature of size 0 (a query node), matching features were already retrieved using the inverted index in the first step. Given a query feature of size 1 (a query sequence), we already have the matching features of size 0 (the query nodes) that are this query feature’s parents. For both these query nodes, we determine the sets of sequences that are their direct children. A sequence in the intersection of these two sets is a match of the query feature, if it connects the nodes in the same manner as the query.
feature (i.e.: has the same source and target). Given a query feature of size $n$ ($n > 1$), which is a basic feature according to Definition 6, we already have the matching features of size $n - 1$ that are this query feature’s parents. For both these parents, we determine the sets of features that are their direct children. A feature in the intersection of these sets is a match of the query feature, if it connects the same nodes as the query feature, connects these nodes in the same manner. Continuing with the example, query $a$ has a basic sequence feature ‘Order’→’Receive’ has two matches in the FNet, ‘order goods’→’receive goods’, and ‘order goods online’→’receive goods’. The feature, ‘receive goods’→’order goods’, is not a match since its source (target) node matches with the target (source) node of the query feature.

Fourthly, after getting the matching features for all basic query features, the mapping between features and graphs maintained in the FNet is used to retrieve and check matching graphs for the query graph, as defined in Definition 8. A
Algorithm 5 presents the algorithm to querying with basic features through the indexes (step 3). For a basic node, matching node features are retrieved through an inverted index (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 an FNet. 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 structure of the query feature (line 1-7). 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 ≤ i ≤ 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.

6 Implementation

Section 5 presents the techniques for advanced process querying using the process feature index (FNet). This chapter describes the architecture that we propose for implementing an FNet. The architecture is based on the more general architecture for business process model repositories proposed in [22] and focuses on the process querying 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 prototype of the architecture and the techniques.

The remainder of the section presents the general layered architecture of the tool in terms of a component diagram; makes the architecture more concrete, by presenting details of the interfaces of the components in terms of class diagrams; and presents the prototype that implements the architecture.
Algorithm 5: Basic Feature Retrieval

1 function checkEdge (qf, pqf, f, M)
2 begin
3 foreach (qn1, qn2) ∈ (E_{qf} - E_{pqf}) do
4     foreach n1 ∈ (M(qn1) ∩ N_f) do
5         foreach n2 ∈ (M(qn2) ∩ N_f) do
6             if (n1, n2) ∈ E_f then return True;
7 return False;

input: a query feature: qf, an FNet: FNet = (F, RF, υ),
       a threshold: max, a mapping: M : Γ(qg, max) → P(⋃g∈D Γ(g, max))
       //Γ(g, max) is defined in Definition 5
output: a set of features: RS, the mapping: M

8 begin
9 if ∃(qf, F) ∈ M then
10     return M(qf); // F ∈ ∪g∈D Γ(g, max) is a set of features
11 else
12     if |E_{qf}| == 0 then
13         if ∀n ∈ N_f, Θ(n) = wildcard then
14             RS ← {f ∈ ∪g∈D Γ(g, max)|||N_f|| = 1}; //all node features in F
15         else
16             RS is the set of matching node features of qf according to
17             Definition 4;
18     else if |E_{qf}| > 0 & ∀e ∈ E_{qf}, θ(n) = basic then
19         RS ← F;
20     foreach pqf ∈ DPFS(qf) do
21        PRS ← featureRetrieval (pqf, FNet); // recursion
22         RS ← RS ∩ (∪f∈PRS {f ∈ RF(pqf)|checkEdge(qf, pqf, f, M) == True});
23 end
24 M(qf) = RS;
25 return RS;

6.1 Component Diagram

Fig. 7 presents the general architecture of the tool, which is based on the reference architecture presented in [22]. 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 paper presents a detailed architecture for the querying function only.

The architecture consists of four layers: the presentation layer, the process repository management layer, the DBMS layer, and the storage layer. The presentation layer provides a (graphical) interface for users to interact with the tool. The process repository management layer provides process specific management,
including functions to construct and manage indexes, retrieve through indexes, and transform process model from the external to internal format. The DBMS layer provides functionality of a database management system, i.e., access and transaction management. The storage layer provide data management (create, update, read, and delete) process data stored in the repository, including indexing information, an internal representation of the business process models that focuses on performance and an external representation of the business process models that focuses on interoperability.

Although a business process model repository would typically contain a number of model management functions (such as checking in and checking out models, version management and configuration management), this chapter focuses only on the retrieval function. However, the implementation of the querying function is consistent with the possible implementation of other functions. The
querying function is implemented by a single component, such that additional components that implement other functions can be added easily.

6.2 Class Diagrams

Fig. 8 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 trans-
forming a given process model into a process graph (Definition 1) and “getFeatures” for deriving the features of a given process graph (Definition 5 and 6). 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 FNet (Algorithm 1); it provides operations “insertGraph”, “updateGraph”, and “deleteGraph” that inserts, updates and deletes a process graph in an FNet (Algorithms 2, 4, and 3). The component also provides an operation for querying process features through the indexes, “queryFeatures”, which returns, given a basic feature, the matching features (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 8). The retrieval component provides one operation, “query”, which retrieves the models that satisfy a given model is supported by operations “queryFeatures” and “check” of the index management component (Section 5).

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. 8 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, “FNet”, 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. 8 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.

6.3 Prototype

As a proof of concept, we implemented a 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 be stored in the repository.

7 Synthetic Process Model Generator

To evaluate the scalability of the process querying technique, 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. The generation of synthetic labels is performed 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. The generation of synthetic process models is performed by decomposing original process models into features (Definition 5), 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 retain 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 a collection that are relevant to the generation of synthetic models in the context of the querying techniques as we proposed. Secondly, it presents an algorithm to generate a collection of synthetic process models based on these properties.

7.1 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 querying technique. The two most common aspects of business process models [22], 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 consi-
der 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 10
presents the word set of a process model collection.

**Definition 10 (Word Set, Label Size).**

Let \( D \) 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 \( W \) of the collection \( D \) is the set of words appear in \( \mathcal{L} \). Formally,
\[
W = \{ w | w \in \omega(l) \land l \in \mathcal{L} \}.
\]

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 9 is \{order, goods, receive,
online, pay, application, approve\}.

![Fig. 9. A Collection of Business Process Graphs](image_url)

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 11 (Frequency of Word Occurrence, Probability of Word
Occurrence).** Let \( D \) be a collection of process graphs with disjoint sets of nodes,
let \( \mathcal{N} \) 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 \( \text{FWO}(w) \), is the
number of nodes in the collection that contain the word \( w \) in their labels. Formally,
\[
\text{FWO}(w) = |\{ n \in \mathcal{N} | w \in \omega(\lambda(n)) \}|.
\]

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,
\[
\text{PWO}(w) = \frac{\text{FWO}(w)}{\sum_{w \in W} \text{FWO}(w)}.
\]
For example, in Figure 9, 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 12, 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 9, 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’.

The frequencies of three types of word co-occurrence are defined as follows.

**Definition 12 (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, $\text{FWCO}(w, w_1) = |\{n \in \mathcal{N} | w, w_1 \in \omega(\lambda(n))\}|$. 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$_{\text{pre}}$): 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, $\text{FWCO}_{\text{pre}}(w, w_1) = |\{n_1 \in \mathcal{N} | g \in \mathcal{D} \land n, n_1 \in N_g \land n_1 \in \bullet n \land w \in \omega(\lambda(n)) \land w_1 \in \omega(\lambda(n_1))\}|$. We say that word $w_1$ co-occurs with $w$ in a pre-set node label if $\text{FWCO}_{\text{pre}}(w, w_1) > 0$.
- Frequency of Post-Word Co-Occurrence (FWCO$_{\text{post}}$): 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, $\text{FWCO}_{\text{post}}(w, w_1) = |\{n_1 \in \mathcal{N} | g \in \mathcal{D} \land n, n_1 \in N_g \land n_1 \in \bullet n \land w \in \omega(\lambda(n)) \land w_1 \in \omega(\lambda(n_1))\}|$. We say that word $w_1$ co-occurs with $w$ in a post-set node label if $\text{FWCO}_{\text{post}}(w, w_1) > 0$.

For example, with respect to the word ‘receive’ in the collection in Figure 9, 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 13.

**Definition 13 (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 $\mathcal{D}$,
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,

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

(1)

- Probability of Pre Word Co-Occurrences (PWCO\text{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,

$$PWCO_{\text{pre}}(w, w_1) = \frac{FWCO_{\text{pre}}(w, w_1)}{\sum_{w_2 \in W} FWCO_{\text{pre}}(w, w_2)}.$$  

(2)

- Probability of Post Word Co-Occurrences (PWCO\text{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,

$$PWCO_{\text{post}}(w, w_1) = \frac{FWCO_{\text{post}}(w, w_1)}{\sum_{w_2 \in W} FWCO_{\text{post}}(w, w_2)}.$$  

(3)

For example, with respect to the word ‘receive’ in the collection in Figure 9, 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 paper. In this section, properties of these features are considered as structural properties, e.g., feature type and feature size as defined in Definition 5. Furthermore, the composition rules are abstracted, which indicate how features can be composed to form a process graph.

As defined in Definition 5, four types of structural features are considered for the purpose of process querying, i.e., sequence, split, join, and loop. Process graphs consist of compositions of these features. For example, graph 6 of Figure 10 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.

When splitting a process graph into features, a feature can be a parent of different features (parent and child features are defined in Definition 7). For example, in graph 6 of Figure 10, 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 14. 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 14 (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 $f \in \text{LMF}(g)$, 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 7.2 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 15 (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 $n$ in the feature is an 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 10, 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.

![Fig. 10. A Business Process Graph](image_url)
for join feature \{c, d\} → h. When generating a synthetic graph, given \( b \to \{c, d, e, f, g\} \) 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 7.2.

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 10, 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 16.

**Definition 16 (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 \( \zeta(N'') = \text{Im} \); 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 \( g \), \( n_1 \to \ldots \to n_2 \), we say that \( f \) has an eventual join, denoted as \( \zeta(N'') = \text{Ev} \). \( \text{Im} \) and \( \text{Ev} \) are the type of a split (join) association of open nodes. The join association of open nodes is defined similarly.

For example, Figure 11 shows some split and join features of graph 6 in Figure 10, the split feature \( b \to \{c, d, e, f, g\} \) has an immediate join \( \{c, d\} \) and an eventual join \( \{d, e\} \); while the join feature \( \{c, d\} \to h \) has an immediate split \( \{c, d\} \) and the join feature \( \{d, i\} \to j \) has an eventual split \( \{d, i\} \). For an immediate join (split), a join (split) feature is inserted to extend the process graph; for an eventual join (split), a join (split) feature and several sequence features are inserted to extend the process graph.

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 17 (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.
- $\text{preON} = \{ n \in N | (n - N) \neq \emptyset \}$ is the pre-open node set.
- $\text{postON} = \{ n \in N | (n \bullet - N) \neq \emptyset \}$ 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 16.

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 18 presents how to abstract all components from a process graph or a collection of process graphs.

**Definition 18 (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 19.

**Definition 19 (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 $M : N_1 \rightarrow N_2$, 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, $\forall sN_1 \in \text{postON}_1(\text{preON}_1): \zeta_1(sN_1) = \zeta_2(M(sN_1)) | n_1 \in sN_1$. 

---

**Fig. 11.** Examples of Join and Split Associations of Open Nodes of *graph 6* in Figure 10

Legend: ○ represents a open node; ■ represents that the open nodes in it are associated.
For example, in Figure 11, components \{c, d\} \rightarrow h and \{d, i\} \rightarrow 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 20 (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 \(FCO(c, C) = |\{f \in \Gamma(g) | g \in D \land f \in LMF(g) \land Comp(f, g) = c\}|\); 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 \(PCO(c, C) = \frac{FCO(c, C)}{\sum_{c_1 \in C} FCO(c_1, C)}\).

### 7.2 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 21.

**Definition 21 (Synthetic Label).** Let \(D\) be a collection of process graphs with disjoint sets of nodes, \(N\) be the node set of \(D\), \(W\) be the word set of \(D\), 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, \(w \in W\), and a set of \(s - 1\) words, \(W = \{w_1, w_2, ..., w_{s-1}\} \subseteq W (w \notin W)\), which co-occur with \(w\).

The probability of the size \(s\) is \(\frac{|\{n \in N | \omega(\lambda(n)) = s\}|}{|N|}\). The probability of selecting a word \(w\) is \(PWO(w)\), according to Definition 11. The probability of selecting a word \(w_i \in W\) is \(PWCO(w, w_i)\), according to Definition 12.

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 22 (Probability of Label Selection).**

Let \(SL\) a set of synthetic labels generated according to Definition 21, let \(sl\) be a synthetic, \(sl \in SL\), and let \(W = \{w | w \in \omega(sl) \land sl \in SL\}\) be the word set of \(SL\). Let \(c\) be a component and let \(n\) be a node of \(c, n \in N_c\).

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 \(w\).
\[ PLS(w) = \begin{cases} \frac{\sum_{w' \in W_{post}} FWCO_{pre}(w', w) + \sum_{w' \in W_{pre}} FWCO_{post}(w', w)}{\sum_{w'' \in W} \left( \sum_{w' \in W_{post}} FWCO_{pre}(w', w'') + \sum_{w' \in W_{pre}} FWCO_{post}(w', w'') \right)}, \\ \text{if } \sum_{w'' \in W} \left( \sum_{w' \in W_{post}} FWCO_{pre}(w', w'') + \sum_{w' \in W_{pre}} FWCO_{post}(w', w'') \right) \neq 0; \\ PW0(w), \text{ otherwise.} \end{cases} \]

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

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 23 (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 6 presents the algorithm for generating synthetic process graphs. Firstly, a set of synthetic labels are generated, according to Definition 21, 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 7. 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 explained in Definition 20. Finally, the open nodes in the join (split) associations are merged as explained in Algorithm 8.

Algorithm 7 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
Algorithm 6: Synthetic Process Graph Generation

**input**: a collection of process graphs: $D$, an integer: $size_c$

**output**: a collection of synthetic process graphs $SD$

```
begin
    $SL$ is the set of generated synthetic labels (Definition 21);
    $SD ← ∅$;
    $C$ is the component set of $D$ (Definition 18);
    while $|SD| < size_c$ do
        $sg = \{\{sn\}, \emptyset, \emptyset, \{sn\}, \emptyset\}; // sn is a newly created node.$
        select $size_g ∈ \{|E_g| | g ∈ D\}$ with probability $PGS(size_g, D)$ (Definition 23);
        while $|E_{sg}| < size_g ∧ (preON_{sg} ∪ postON_{sg}) ≠ ∅$ do
            randomly select $n ∈ preON_{sg} ∪ postON_{sg}$;
            if $n ∈ postON_{sg}$ then
                if $\not\exists N ⊆ N_{sg} : n ∈ N ∧ \zeta_{sg}(N) ∈ \{Im, Ev\}$ then
                    $C_1 ← \{c ∈ C | preON_{c} ≠ ∅\};$
                    select $c ∈ C_1$ with probability $PCO(c, C_1)$ (Definition 20);
                    randomly select $n_1 ∈ preON_{c};$
                    $sg ← mergeNode(sg, c, n, n_1, SL);$ //Algorithm 7.
                else
                    randomly select $AN ∈ \{N ⊆ N_{sg} | n ∈ N ∧ \zeta_{sg}(N) ∈ \{Im, Ev\}\};$
                    $C_1 ← \{c ∈ C | Type(c) = join ∧ AN ⊆ preON_{c} ∧ |AN| = |AN_1| ∧ \zeta_{sg}(AN) = \zeta_{sg}(AN_1)\};$
                    select $c ∈ C_1$ with probability $PCO(c, C_1)$ (Definition 20);
                    $sg ← mergeAsso(sg, c, AN, AN_1, SL);$ //Algorithm 8.
            else
                if $n ∈ preON_{sg}$ then //similar to lines 11-20.
                    $SD ← SD ∪ \{sg\};$
            return $SD$.
end
```

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 22.

Algorithm 8 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 7). 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 7: Merge a Pair of Open Nodes

\textbf{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: \(\mathcal{S}\mathcal{L}\)

\textbf{output} : a synthetic process graphs: \(sg\)

\begin{algorithmic}[1]
\State \(N_{sg} \leftarrow N_{sg} \cup (N_c - \{n_c\})\);
\If {\(n_{sg} \in \text{preON}_{sg} \land n_c \in \text{postON}_c\)}
\State \(\text{preON}_{sg} \leftarrow (\text{preON}_{sg} - \{n_{sg}\}) \cup \text{preON}_c\);
\State \(\text{postON}_{sg} \leftarrow \text{postON}_{sg} \cup (\text{postON}_c - \{n_c\})\);
\EndIf
\ElseIf {\(n_{sg} \in \text{postON}_{sg} \land n_c \in \text{preON}_c\)}
\State \(\text{preON}_{sg} \leftarrow \text{preON}_{sg} \cup (\text{preON}_c - \{n_c\})\);
\State \(\text{postON}_{sg} \leftarrow \text{postON}_{sg} \cup \text{postON}_c\);
\EndIf
\State \(E_{sg} \leftarrow E_{sg} \cup \{(n_3,n_3) | n_3 \in \bullet \} \cup \{(n_3,n_3) | n_3 \in \bullet \}\);
\State \(\lambda_{sg} \leftarrow \lambda_{sg} \cup \{(n,sl)\}\);
\Return \(sg\);
\end{algorithmic}

Algorithm 8: Merge Join and Split Associations of Open Nodes

\textbf{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: \(\mathcal{S}\mathcal{L}\)

\textbf{output} : a synthetic process graphs: \(sg\)

\begin{algorithmic}[1]
\Begin
\For {\(n_{sg} \in AN_{sg}\)}
\State randomly select \(n_c\) from \(AN_c\);
\If {\(\zeta_{sg}(AN_{sg}) = \text{Im}\)}
\State \(sg \leftarrow \text{mergeNode}(sg,c,n_{sg},n_c,\mathcal{S}\mathcal{L});\) //Algorithm 7.
\EndIf
\ElseIf {\(\zeta_{sg}(AN_{sg}) = \text{Ev}\)}
\State \(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}) \land n_1 \rightarrow \ldots \rightarrow n_2 \text{ is a subgraph of } c_1)\};
\State select \(c_1 \in C_1\) with probability \(\text{FCO}(c_1, C_1)\) (Definition 20);
\State \(sg \leftarrow \text{mergeNode}(sg,c_1,n_{sg},n_1,\mathcal{S}\mathcal{L});\) //Algorithm 7.
\EndIf
\State \(sg \leftarrow \text{mergeNode}(sg,c,n_2,n_c,\mathcal{S}\mathcal{L});\) //Algorithm 7.
\EndFor
\Delete \(AN_{sg}\) from the domain of \(\zeta_{sg}\);
\Delete \(AN_c\) from the domain of \(\zeta_c\);
\For {\(AN_1 \in \{N_1 \subseteq N_c | \zeta_c(N_1) \in \{\text{Im}, \text{Ev}\}\}\)}
\State \(\zeta_{sg}(AN_1) \leftarrow \zeta_c(AN_1)\);
\EndFor
\Return \(sg\);
\End
\end{algorithmic}
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.

8 Evaluation

This section shows how the use of the technique in this paper affects process querying in terms of performance and quality. Two types of business process model collections were used in the experiments, one real-life collections and two synthetic collections. The real-life collection consists of 604 SAP reference models [14]. 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 synthetic process model on average contains 20.3 nodes and 24.1 edges.

8.1 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 1. In the first group, as shown in Figure 12, five queries were adapted from the evaluation of BPMN-Q [1]. Query a, b, d, e, and g in the evaluation of [1] were selected. Query c and f were not selected, because they test specific types of elements in BPMN-Q that are simply considered nodes in this paper 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.

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
is composed of 9 basic nodes and 8 basic edges; query 7 contains 1 wildcard nodes 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 1. Properties of Querying with SAP reference models

<table>
<thead>
<tr>
<th>Query Name</th>
<th>Basic Elements</th>
<th>Advanced Elements</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</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>Group 2</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>

Fig. 12. First Group of Queries

Evaluation Results To evaluate the performance of the technique, both groups of queries were used to run experiments. Table 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 an FNet. 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 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 1).

Table 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.

To better evaluate the performance of our technique, we compared it with the performance of BPMN-Q [2]. 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 an FNet 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 paper have similar characteristics as in [2] 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 have one positive result within the two models. Second, for each query, three models were artificially made to check if the technique in this paper 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.

8.2 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 7. 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 8.1. For each query models in the first group of query models in Section 8.1, 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 paper 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 paper.

Evaluation Results Table 3 shows the execution time of the process querying running with 604 synthetic models. The structure of the table is the same as Table 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 FNet 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 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
Table 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>0.01s</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.

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 (FNet) should be better than a linear algorithm. Using BPMN-Q [2], 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.

From Table 3 and 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 5. 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 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>0.10s</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.

9 Related Work

The work presented in this paper is related to business process querying, business process similarity search, general graph querying and general model querying.

Three groups of researchers have been working on advanced business process querying [2,3,6]. Awad [2] develops BPMN-Q, a language to query business processes, by extending the BPMN notation and implements BPMN-Q on top of
Table 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>

relational databases. Beeri et al. [3] propose BP-QL, a language to query business processes modeled in BPEL. Choi et al. [6] propose IPM-EPDL, a query language for a proprietary process modeling notation based on XML. The difference between this paper and the above work is that this paper focuses on developing indexing techniques to make advanced business process querying more efficient. Jin et al. [9] develop efficient indexing techniques for basic business process querying, using sequences in the process models. The differences between this paper and [9] are as follows. Firstly, the technique in this paper supports advanced business process querying besides basic business process querying. Secondly, more features besides sequences are evaluated in this paper. Besides the work on querying business process models, there also exists work on querying executions of business processes [4,5].

The technique in this paper relates to the topic of business process similarity search. Process querying and similarity partly share the same techniques, e.g., for node matching. Similarity search is used to retrieve process models that are similar to a given query model instead of exactly matching and thus uses different techniques to perform the search. The feature-based indexing has also been applied on business process similarity search to improve the efficiency of similarity search [21,23]. The main difference between this paper and [21,23] is that the metrics for matching (features of) process models are different because of the differences of process similarity search and querying, therefore retrieving features through the indexes are also different. In addition to that, features used in this paper are different, i.e., the loop feature is used for process querying, while the start and stop features are used for process similarity search. Other work on improving the efficiency of similarity search includes the work by Kunze et al. [11], who propose a metric that enables the use of an MTree index on process models. Qiao et al. [13] use clustering techniques to search business process models efficiently.

General graph querying has been applied in various application domains, including fingerprint, DNA and chemical compound search. Willett et al. [19] describe a feature-based similarity search algorithm for searching in a chemical compound databases. ShaSha et al. [16] propose a path-based approach; Yan et al. [20] use discriminative frequent structures to index graphs. The main difference between the work that has been done in this area and the work in this paper is the different nature of business process graphs as compared to 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. The selection of features is also different. In this paper common workflow patterns are used due to the characteristics of process models.

General model querying also relates to the topic of business process similarity search. Query languages [17,18] have been designed based on UML (e.g., class diagrams) instead of process modeling notations (e.g., BPMN and EPCs). Therefore, these query languages are for querying general software models instead of process models.

### 10 Conclusion

This paper presents a technique for improving the efficiency of advanced business process querying, which can be used to efficiently retrieve specific business models from the large sets of business process models that we encounter nowadays in practice. The technique works by breaking up a process model into small sub-models, which can also be used to build an index, called a feature net. In particular, the technique can also deal with advanced querying structures, such as paths of edges.

Experiments show that a feature net can be used to retrieve results two orders of magnitude faster than querying techniques that is built on top of traditional RDBMS [2].

There are several directions for improving the feature net. Firstly, the technique in this paper focuses on tasks and relations between tasks. However, process models often contain other information that may be exploited, e.g., resources. This information can be integrated into an FNet by add more dimensions into features, e.g., resources and relations between resources and tasks. Querying on the basis of this information is left for future work. Secondly, label matching is based on matching identical words. However, equivalent tasks can be labeled differently, e.g., due to the use synonyms and different levels of verbosity. Therefore, we applied more advanced metrics for label matching that consider synonyms [7] and domain ontologies [10]. The integration of these advanced metrics into the technique described in this paper is also left for future work.

**Acknowledgement** - The research presented in this paper is supported by the China Scholarship Council (CSC).

**References**

<table>
<thead>
<tr>
<th>nr.</th>
<th>Year</th>
<th>Title</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>385</td>
<td>2012</td>
<td>FNet: An Index for Advanced Business Process Querying</td>
<td>Zhiqiang Yan, Remco Dijkman, Paul Grefen</td>
</tr>
<tr>
<td>384</td>
<td>2012</td>
<td>Defining Various Pathway Terms</td>
<td>W.R. Dalinghaus, P.M.E. Van Gorp</td>
</tr>
<tr>
<td>383</td>
<td>2012</td>
<td>The Service Dominant Strategy Canvas: Defining and Visualizing a Service Dominant Strategy through the Traditional Strategic Lens</td>
<td>Egon Lüftenegger, Paul Grefen, Caren Weisleder</td>
</tr>
<tr>
<td>382</td>
<td>2012</td>
<td>A Stochastic Variable Size Bin Packing Problem With Time Constraints</td>
<td>Stefano Fazi, Tom van Woensel, Jan C. Fransoo</td>
</tr>
<tr>
<td>381</td>
<td>2012</td>
<td>Coordination and Analysis of Barge Container Hinterland Networks</td>
<td>K. Sharypova, T. van Woensel, J.C. Fransoo</td>
</tr>
<tr>
<td>380</td>
<td>2012</td>
<td>Proximity matters: Synergies through co-location of logistics establishments</td>
<td>Frank P. van den Heuvel, Peter W. de Langen, Karel H. van Donselaar, Jan C. Fransoo</td>
</tr>
<tr>
<td>379</td>
<td>2012</td>
<td>A literature review in process harmonization: a conceptual framework</td>
<td>Heidi Romero, Remco Dijkman, Paul Grefen, Arjan van Weele</td>
</tr>
<tr>
<td>376</td>
<td>2012</td>
<td>Improving the performance of sorter systems by scheduling inbound containers</td>
<td>K. Fikse, S.W.A. Haneyah, J.M.J. Schutten</td>
</tr>
<tr>
<td>375</td>
<td>2012</td>
<td>Strategies for dynamic appointment making by container terminals</td>
<td>Albert Douma, Martijn Mes</td>
</tr>
<tr>
<td>374</td>
<td>2012</td>
<td>MyPHRMachines: Lifelong Personal Health Records in the Cloud</td>
<td>Pieter van Gorp, Marco Comuzzi</td>
</tr>
<tr>
<td>373</td>
<td>2012</td>
<td>Service differentiation in spare parts supply through dedicated stocks</td>
<td>E.M. Alvarez, M.C. van der Heijden, W.H.M. Zijm</td>
</tr>
<tr>
<td>Year</td>
<td>Title</td>
<td>Authors</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td><strong>Spare parts inventory pooling: how to share the benefits</strong></td>
<td>Frank Karsten, Rob Basten</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td><strong>Condition based spare parts supply</strong></td>
<td>X.Lin, R.J.I. Basten, A.A. Kranenburg, G.J. van Houtum</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td><strong>Using Simulation to Assess the Opportunities of Dynamic Waste Collection</strong></td>
<td>Martijn Mes</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td><strong>Aggregate overhaul and supply chain planning for rotables</strong></td>
<td>J. Arts, S.D. Flapper, K. Vernooij</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td><strong>Operating Room Rescheduling</strong></td>
<td>J.T. van Essen, J.L. Hurink, W. Hartholt, B.J. van den Akker</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td><strong>Switching Transport Modes to Meet Voluntary Carbon Emission Targets</strong></td>
<td>Kristel M.R. Hoen, Tarkan Tan, Jan C. Fransoo, Geert-Jan van Houtum</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td><strong>On two-echelon inventory systems with Poisson demand and lost sales</strong></td>
<td>Elisa Alvarez, Matthieu van der Heijden</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td><strong>Minimizing the Waiting Time for Emergency Surgery</strong></td>
<td>J.T. van Essen, E.W. Hans, J.L. Hurink, A. Oversberg</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td><strong>Vehicle Routing Problem with Stochastic Travel Times Including Soft Time Windows and Service Costs</strong></td>
<td>Duygu Tas, Nico Dellaert, Tom van Woensel, Ton de Kok</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td><strong>A New Approximate Evaluation Method for Two-Echelon Inventory Systems with Emergency Shipments</strong></td>
<td>Erhun Özkan, Geert-Jan van Houtum, Yasemin Serin</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td><strong>Approximating Multi-Objective Time-Dependent Optimization Problems</strong></td>
<td>Said Dabia, El-Ghazali Talbi, Tom Van Woensel, Ton de Kok</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td><strong>Branch and Cut and Price for the Time Dependent Vehicle Routing Problem with Time Window</strong></td>
<td>Said Dabia, Stefan Röpke, Tom Van Woensel, Ton de Kok</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td><strong>Analysis of an Assemble-to-Order System with Different Review Periods</strong></td>
<td>A.G. Karaarslan, G.P. Kiesmüller, A.G. de Kok</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td><strong>Interval Availability Analysis of a Two-Echelon, Multi-Item System</strong></td>
<td>Ahmad Al Hanbali, Matthieu van der Heijden</td>
<td></td>
</tr>
<tr>
<td>Page</td>
<td>Title</td>
<td>Authors</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>358</td>
<td><strong>Carbon-Optimal and Carbon-Neutral Supply Chains</strong></td>
<td>Felipe Caro, Charles J. Corbett, Tarkan Tan, Rob Zuidwijk</td>
<td></td>
</tr>
<tr>
<td>357</td>
<td><strong>Generic Planning and Control of Automated Material Handling Systems: Practical Requirements Versus Existing Theory</strong></td>
<td>Sameh Haneyah, Henk Zijm, Marco Schutten, Peter Schuur</td>
<td></td>
</tr>
<tr>
<td>356</td>
<td><strong>Last time buy decisions for products sold under warranty</strong></td>
<td>M. van der Heijden, B. Iskandar</td>
<td></td>
</tr>
<tr>
<td>355</td>
<td><strong>Spatial concentration and location dynamics in logistics: the case of a Dutch provence</strong></td>
<td>Frank P. van den Heuvel, Peter W. de Langen, Karel H. van Donselaar, Jan C. Fransoo</td>
<td></td>
</tr>
<tr>
<td>354</td>
<td><strong>Identification of Employment Concentration Areas</strong></td>
<td>Frank P. van den Heuvel, Peter W. de Langen, Karel H. van Donselaar, Jan C. Fransoo</td>
<td></td>
</tr>
<tr>
<td>353</td>
<td><strong>BOMN 2.0 Execution Semantics Formalized as Graph Rewrite Rules: extended version</strong></td>
<td>Pieter van Gorp, Remco Dijkman</td>
<td></td>
</tr>
<tr>
<td>352</td>
<td><strong>Resource pooling and cost allocation among independent service providers</strong></td>
<td>Frank Karsten, Marco Slikker, Geert-Jan van Houtum</td>
<td></td>
</tr>
<tr>
<td>351</td>
<td><strong>A Framework for Business Innovation Directions</strong></td>
<td>E. Lüftenegger, S. Angelov, P. Grefen</td>
<td></td>
</tr>
<tr>
<td>350</td>
<td><strong>The Road to a Business Process Architecture: An Overview of Approaches and their Use</strong></td>
<td>Remco Dijkman, Irene Vanderfeesten, Hajo A. Reijers</td>
<td></td>
</tr>
<tr>
<td>349</td>
<td><strong>Effect of carbon emission regulations on transport mode selection under stochastic demand</strong></td>
<td>K.M.R. Hoen, T. Tan, J.C. Fransoo, G.J. van Houtum</td>
<td></td>
</tr>
<tr>
<td>348</td>
<td><strong>An improved MIP-based combinatorial approach for a multi-skill workforce scheduling problem</strong></td>
<td>Murat Fırat, Cor Hurkens</td>
<td></td>
</tr>
<tr>
<td>347</td>
<td><strong>An approximate approach for the joint problem of level of repair analysis and spare parts stocking</strong></td>
<td>R.J.I. Basten, M.C. van der Heijden, J.M.J. Schutten</td>
<td></td>
</tr>
<tr>
<td>346</td>
<td><strong>Joint optimization of level of repair analysis and spare parts stocks</strong></td>
<td>R.J.I. Basten, M.C. van der Heijden, J.M.J. Schutten</td>
<td></td>
</tr>
<tr>
<td>345</td>
<td><strong>Inventory control with manufacturing lead time flexibility</strong></td>
<td>Ton G. de Kok</td>
<td></td>
</tr>
<tr>
<td>344</td>
<td><strong>Analysis of resource pooling games via a new extension of the Erlang loss function</strong></td>
<td>Frank Karsten, Marco Slikker, Geert-Jan van Houtum</td>
<td></td>
</tr>
<tr>
<td>343</td>
<td><strong>Vehicle refueling with limited resources</strong></td>
<td>Murat Fırat, C.A.J. Hurkens, Gerhard J. Woeginger</td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>Title</td>
<td>Authors</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>Optimal Inventory Policies with Non-stationary Supply Disruptions and Advance Supply Information</td>
<td>Bilge Atasoy, Refik Güllü, Tarkan Tan</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>Redundancy Optimization for Critical Components in High-Availability Capital Goods</td>
<td>Kurtulus Baris Öner, Alan Scheller-Wolf, Geert-Jan van Houtum</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>Analysis of a two-echelon inventory system with two supply modes</td>
<td>Joachim Arts, Gudrun Kiesmüller</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>Analysis of the dial-a-ride problem of Hunsaker and Savelsbergh</td>
<td>Murat Fırat, Gerhard J. Woeginger</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>Attaining stability in multi-skill workforce scheduling</td>
<td>Murat Fırat, Cor Hurkens</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>An exact approach for relating recovering surgical patient workload to the master surgical schedule</td>
<td>P.T. Vanberkel, R.J. Boucherie, E.W. Hans, J.L. Hurink, W.A.M. van Lent, W.H. van Harten</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>Efficiency evaluation for pooling resources in health care</td>
<td>Peter T. Vanberkel, Richard J. Boucherie, Erwin W. Hans, Johann L. Hurink, Nelly Litvak</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>The Effect of Workload Constraints in Mathematical Programming Models for Production Planning</td>
<td>M.M. Jansen, A.G. de Kok, I.J.B.F. Adan</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>Using pipeline information in a multi-echelon spare parts inventory system</td>
<td>Christian Howard, Ingrid Reijnen, Johan Marklund, Tarkan Tan</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>Reducing costs of repairable spare parts supply systems via dynamic scheduling</td>
<td>H.G.H. Tiemessen, G.J. van Houtum</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>A combinatorial approach to multi-skill workforce scheduling</td>
<td>Murat Fırat, Cor Hurkens</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>Stability in multi-skill workforce scheduling</td>
<td>Murat Fırat, Cor Hurkens, Alexandre Laugier</td>
<td></td>
</tr>
</tbody>
</table>
Maintenance spare parts planning and control: A framework for control and agenda for future research


Near-optimal heuristics to set base stock levels in a two-echelon distribution network

R.J.I. Basten, G.J. van Houtum

Inventory reduction in spare part networks by selective throughput time reduction

M.C. van der Heijden, E.M. Alvarez, J.M.J. Schutten

The selective use of emergency shipments for service-contract differentiation

E.M. Alvarez, M.C. van der Heijden, W.H. Zijn

Heuristics for Multi-Item Two-Echelon Spare Parts Inventory Control Problem with Batch Ordering in the Central Warehouse

B. Walrave, K. v. Oorschot, A.G.L. Romme

Preventing or escaping the suppression mechanism: intervention conditions

Nico Dellaert, Jully Jeunet.

Hospital admission planning to optimize major resources utilization under uncertainty

R. Seguel, R. Eshuis, P. Grefen.

Minimal Protocol Adaptors for Interacting Services


Teaching Retail Operations in Business and Engineering Schools

Lydie P.M. Smets, Geert-Jan van Houtum, Fred Langerak.

Design for Availability: Creating Value for Manufacturers and Customers

Pieter van Gorp, Rik Eshuis.

Transforming Process Models: executable rewrite rules versus a formalized Java program

Bob Walrave, Kim E. van Oorschot, A. Georges L. Romme

Getting trapped in the suppression of exploration: A simulation model

S. Dabia, T. van Woensel, A.G. de Kok


2010

Tales of a So(u)rcerer: Optimal Sourcing

Osman Alp, Tarkan Tan
<table>
<thead>
<tr>
<th>Page</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>311</td>
<td><strong>Decisions Under Alternative Capacitated Suppliers and General Cost Structures</strong></td>
<td>R.A.C.M. Broekmeulen, C.H.M. Bakx</td>
</tr>
<tr>
<td>310</td>
<td><strong>In-store replenishment procedures for perishable inventory in a retail environment with handling costs and storage constraints</strong></td>
<td>E. Lüftenegger, S. Angelov, E. van der Linden, P. Grefen</td>
</tr>
<tr>
<td>309</td>
<td><strong>The state of the art of innovation-driven business models in the financial services industry</strong></td>
<td>R. Seguel, P. Grefen, R. Eshuis</td>
</tr>
<tr>
<td>308</td>
<td><strong>Design of Complex Architectures Using a Three Dimension Approach: the CrossWork Case</strong></td>
<td>K.M.R. Hoen, T. Tan, J.C. Fransoo, G.J. van Houtum</td>
</tr>
<tr>
<td>307</td>
<td><strong>Effect of carbon emission regulations on transport mode selection in supply chains</strong></td>
<td>Martijn Mes, Matthieu van der Heijden, Peter Schuur</td>
</tr>
<tr>
<td>306</td>
<td><strong>Interaction between intelligent agent strategies for real-time transportation planning</strong></td>
<td>Marco Slikker, Peter Borm, René van den Brink</td>
</tr>
<tr>
<td>304</td>
<td><strong>Practical extensions to the level of repair analysis</strong></td>
<td>R.J.I. Basten, M.C. van der Heijden, J.M.J. Schutten</td>
</tr>
<tr>
<td>303</td>
<td><strong>Ocean Container Transport: An Underestimated and Critical Link in Global Supply Chain Performance</strong></td>
<td>Jan C. Fransoo, Chung-Yee Lee</td>
</tr>
<tr>
<td>302</td>
<td><strong>Capacity reservation and utilization for a manufacturer with uncertain capacity and demand</strong></td>
<td>Y. Boulaksil; J.C. Fransoo; T. Tan</td>
</tr>
<tr>
<td>300</td>
<td><strong>Spare parts inventory pooling games</strong></td>
<td>F.J.P. Karsten; M. Slikker; G.J. van Houtum</td>
</tr>
<tr>
<td>299</td>
<td><strong>Capacity flexibility allocation in an outsourced supply chain with reservation</strong></td>
<td>Y. Boulaksil, M. Grunow, J.C. Fransoo</td>
</tr>
<tr>
<td>298</td>
<td><strong>An optimal approach for the joint problem of level of repair analysis and spare parts stocking</strong></td>
<td>R.J.I. Basten, M.C. van der Heijden, J.M.J. Schutten</td>
</tr>
<tr>
<td>297</td>
<td><strong>Responding to the Lehman Wave: Sales Forecasting and Supply Management during the Credit Crisis</strong></td>
<td>Robert Peels, Maximiliano Udenio, Jan C. Fransoo, Marcel Wolfs, Tom Hendrikx</td>
</tr>
<tr>
<td>296</td>
<td><strong>An exact approach for relating recovering surgical patient workload to the master surgical schedule</strong></td>
<td>Peter T. Vanberkel, Richard J. Boucherie, Erwin W. Hans, Johann L. Hurink, Wineke A.M. van Lent, Wim H. van Harten</td>
</tr>
<tr>
<td>295</td>
<td><strong>An iterative method for the simultaneous optimization of repair decisions and spare parts stocks</strong></td>
<td>R.J.I. Basten, M.C. van der Heijden, J.M.J. Schutten</td>
</tr>
<tr>
<td>294</td>
<td><strong>Fujaba hits the Wall(-e)</strong></td>
<td>Pieter van Gorp, Ruben Jubeh, Bernhard Grusie, Anne Keller</td>
</tr>
<tr>
<td>292</td>
<td><strong>Business Process Model Repositories -</strong></td>
<td>Zhiqiang Yan, Remco Dijkman, Paul</td>
</tr>
<tr>
<td>Page</td>
<td>Title</td>
<td>Authors</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>291</td>
<td>Efficient Optimization of the Dual-Index Policy Using Markov Chains</td>
<td>Joachim Arts, Marcel van Vuuren, Gudrun Kiesmuller</td>
</tr>
<tr>
<td>290</td>
<td>Hierarchical Knowledge-Gradient for Sequential Sampling</td>
<td>Martijn R.K. Mes; Warren B. Powell; Peter I. Frazier</td>
</tr>
<tr>
<td>289</td>
<td>Analyzing combined vehicle routing and break scheduling from a distributed decision making perspective</td>
<td>C.M. Meyer; A.L. Kok; H. Kopfer; J.M.J. Schutten</td>
</tr>
<tr>
<td>288</td>
<td>Anticipation of lead time performance in Supply Chain Operations Planning</td>
<td>Michiel Jansen; Ton G. de Kok; Jan C. Fransoo</td>
</tr>
<tr>
<td>287</td>
<td>Inventory Models with Lateral Transshipments: A Review</td>
<td>Colin Paterson; Gudrun Kiesmuller; Ruud Teunter; Kevin Glazebro</td>
</tr>
<tr>
<td>286</td>
<td>Efficiency evaluation for pooling resources in health care</td>
<td>P.T. Vanberkel; R.J. Boucherie; E.W. Hans; J.L. Hurink; N. Litvak</td>
</tr>
<tr>
<td>285</td>
<td>A Survey of Health Care Models that Encompass Multiple Departments</td>
<td>P.T. Vanberkel; R.J. Boucherie; E.W. Hans; J.L. Hurink; N. Litvak</td>
</tr>
<tr>
<td>284</td>
<td>Supporting Process Control in Business Collaborations</td>
<td>S. Angelov; K. Vidyasankar; J. Vonk; P. Grefen</td>
</tr>
<tr>
<td>283</td>
<td>Inventory Control with Partial Batch Ordering</td>
<td>O. Alp; W.T. Huh; T. Tan</td>
</tr>
<tr>
<td>282</td>
<td>Translating Safe Petri Nets to Statecharts in a Structure-Preserving Way</td>
<td>R. Eshuis</td>
</tr>
<tr>
<td>281</td>
<td>The link between product data model and process model</td>
<td>J.J.C.L. Vogelaar; H.A. Reijers</td>
</tr>
<tr>
<td>280</td>
<td>Inventory planning for spare parts networks with delivery time requirements</td>
<td>I.C. Reijnen; T. Tan; G.J. van Houtum</td>
</tr>
<tr>
<td>279</td>
<td>Co-Evolution of Demand and Supply under Competition</td>
<td>B. Vermeulen; A.G. de Kok</td>
</tr>
<tr>
<td>278</td>
<td>Toward Meso-level Product-Market Network Indices for Strategic Product Selection and (Re)Design Guidelines over the Product Life-Cycle</td>
<td>B. Vermeulen, A.G. de Kok</td>
</tr>
<tr>
<td>277</td>
<td>An Efficient Method to Construct Minimal Protocol Adaptors</td>
<td>R. Seguel, R. Eshuis, P. Grefen</td>
</tr>
<tr>
<td>276</td>
<td>Coordinating Supply Chains: a Bilevel Programming Approach</td>
<td>Ton G. de Kok, Gabriella Muratore</td>
</tr>
<tr>
<td>275</td>
<td>Inventory redistribution for fashion products under demand parameter update</td>
<td>G.P. Kiesmuller, S. Minner</td>
</tr>
<tr>
<td>274</td>
<td>Comparing Markov chains: Combining aggregation and precedence relations applied to sets of states</td>
<td>A. Busic, I.M.H. Vliegen, A. Scheller-Wolf</td>
</tr>
<tr>
<td>273</td>
<td>Separate tools or tool kits: an exploratory study of engineers’ preferences</td>
<td>I.M.H. Vliegen, P.A.M. Kleingeld, G.J. van Houtum</td>
</tr>
<tr>
<td>272</td>
<td>An Exact Solution Procedure for Multi-Item Two-Echelon Spare Parts Inventory Control Problem</td>
<td>Engin Topan, Z. Pelin Bayindir, Tarkan Tan</td>
</tr>
<tr>
<td>Page</td>
<td>Title</td>
<td>Authors</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>269</td>
<td>Similarity of Business Process Models: Metics and Evaluation</td>
<td>Remco Dijkman, Marlon Dumas, Boudewijn van Dongen, Reina Kaarik, Jan Mendling</td>
</tr>
<tr>
<td>266</td>
<td>Restricted dynamic programming: a flexible framework for solving realistic VRPs</td>
<td>J. Gromicho; J.J. van Hoorn; A.L. Kok; J.M.J. Schutten;</td>
</tr>
</tbody>
</table>

Working Papers published before 2009 see: [http://beta.ieis.tue.nl](http://beta.ieis.tue.nl)