Indexing and Efficient Instance-based Retrieval of Process Models Using Untanglings

Artem Polyvyanyy[1], Marcello La Rosa[1,2], and Arthur H.M. ter Hofstede[1,3]

[1] Queensland University of Technology, Brisbane, Australia
[2] NICTA Queensland Lab, Brisbane, Australia
[3] Eindhoven University of Technology, Eindhoven, The Netherlands
{artem.polyvyanyy;m.larosa;a.terhofstede}@qut.edu.au

Abstract. Process-Aware Information Systems (PAISs) support executions of operational processes that involve people, resources, and software applications on the basis of *process models*. Process models describe vast, often infinite, amounts of *process instances*, i.e., workflows supported by the systems. With the increasing adoption of PAISs, large process model *repositories* emerged in companies and public organizations. These repositories constitute significant information resources. Accurate and efficient retrieval of process models and/or process instances from such repositories is interesting for multiple reasons, e.g., searching for similar models/instances, filtering, reuse, standardization, process compliance checking, verification of formal properties, etc. This paper proposes a technique for *indexing* process models that relies on their alternative representations, called *untanglings*. We show the use of untanglings for retrieval of process models based on process instances that they specify via a solution to the *generalized executability problem*. Experiments with industrial process models testify that the proposed retrieval approach is up to three orders of magnitude faster than the state of the art.

1 Introduction

The Information Systems discipline studies different ways in which information can be processed, often algorithmically using process modeling practices. Workflow management systems, business process management systems, and enterprise information systems are examples of process-aware information systems (PAISs) [1]. PAISs support executions of operational processes on the basis of *process models* that are usually expressed in languages such as the Web Services Business Process Execution Language (WS-BPEL) or the Business Process Model and Notation (BPMN). For example, Fig. 1 shows a BPMN model that describes various scenarios for handling travel quote requests.

Process models describe vast amounts of executions, or *process instances*, for handling similar scenarios. The number of instances that are described in a single process model is *exponential* in the number of decisions that one can take when executing the model and grows *combinatorially* with the amount of tasks that can be executed simultaneously. Moreover, a process model can capture an *infinite* number of executions, in case of loops.

As it becomes increasingly common for organizations to adopt the process-oriented approach to model and execute their routines, organizations often end up managing repositories that comprise up to thousands of process models. For example, Suncorp, the largest Australian insurer, maintains a repository of more than 3,000 models [2,3].

Process model repositories are immense information resources. In order to reduce this *information overload*, one should be striving for automated retrieval systems. Accurate and efficient retrieval of information about process instance that are stored in process model repositories is interesting for several reasons, including:

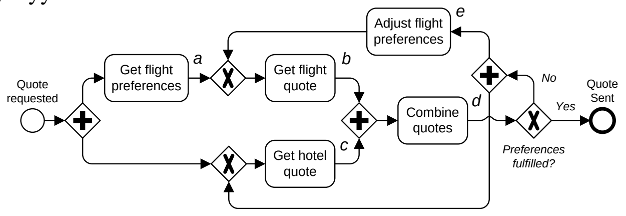


Fig. 1. A BPMN model for handling travel quote requests

- ◦ *Reuse/redesign.* When developing new or modifying existing process models, one can reuse information that is contained in process model repositories [4], e.g., by retrieving process models that specify process instances of interest.
- ◦ *Compliance.* Process models are subject to constraints enforced by regulations and/or laws, often referred to as *compliance rules*. Effective compliance checking requires the retrieval of information about process instances [5].
- ◦ *Standardization.* Standard process models are exemplar models that should be used as references [6]. These models encode best practices for handling similar process instances across several models in a repository. The starting point of a process model standardization initiative often deals with identification of similar process models, i.e., those models that capture identical or similar process instances.

For example, an organization can issue a *compliance rule* which checks that in every travel handling scenario it is never possible that both tasks "*Get flight preferences*" and "*Adjust flight preferences*" occur together. This rule can be triggered to avoid internal adjustments of travel preferences. In this case, the model in Fig. 1 must be retrieved as one that violates the rule. Alternatively, one may want to redesign routines so that every time flight and hotel quotes are processed, there is also an option to propose a quote for renting a car. To implement this intent, one can start by retrieving all models that describe instances in which both tasks "*Get flight quote*" and "*Get hotel quote*" occur.

The contribution of this paper is threefold. First, it proposes an index data structure that is tailored towards efficient retrieval of process models based on information about process instances. The index is due to an alternative representation of process models, called *representative untanglings*. The unique characteristics of this index allow for an unmatched querying experience. Second, it demonstrates this querying experience using *query primitives* that take form of the generalized version of the *classical executability problem* [7]. Given a process model and a set of tasks as input, the *generalized executability problem* deals with deciding if the model describes at least one process instance in which all tasks from the given set occur. Among other applications, a solution to the generalized executability problem can be used to implement the above illustrated retrieval scenarios. Third, it suggests an efficient solution to this problem using representative untanglings. The approach has been implemented. Experiments with industrial models show up to three orders of magnitude speed up compared to the state of the art.

The rest of the paper is organized as follows: Sect. 2 positions our research in the light of related work. Next, Sect. 3 provides preliminary notions. Sect. 4 describes a novel index data structure. Sect. 5 exemplifies the use of this index for querying process model repositories. Sect. 6 reports on the performance measurements of a prototype that implements the developed querying technique. Finally, Sect. 7 concludes the paper.

2 Related Work

Querying deals with retrieving information that is relevant to a given *information need* from a collection of *information resources*. In case of process model querying, information resources are process models (*structural information*) as well as process instances that these process models describe (*behavioral information*).

There exist various techniques to query process model repositories based on structural information, cf. [2,3,8,9,10]. Given a query specified as a structural pattern, or a structural template with wildcards, these techniques are capable of retrieving process models which are formalized as structures that match the pattern, or fit the template. First, indexing techniques are employed to filter the repository by obtaining a set of candidate models that fit the indexed features of the query. Second, graph isomorphism or graph-edit distance techniques [11] are applied to identify the models from the candidate set that score an exact match, or are sufficiently similar, to the query. Differently, we propose a technique that retrieves process models based on behavioral information.

Other techniques retrieve process models based on *abstractions* of behavioral information, cf. [12,13]. They accept loss of behavioral information, and consequently decrease in precision and recall, as the price for efficient retrieval. Our retrieval technique is *precise* and *sensitive*, i.e., it always retrieves *all and only* models that match the query.

Model checking is a technique for verifying various properties of process models [7]. This technique usually proceeds by constructing an alternative representation of a process model and then uses this representation for efficient validation of properties. Model checking can be used to implement process model retrieval that is based on behavioral information as well as is precise and sensitive. Indeed, behavioral information needs can be expressed as properties to be verified. Similarly, our technique makes retrieval decisions based on alternative representations of process models, called representative untanglings. Differently, once constructed, representative untanglings can be reused much more often than artifacts employed for model checking purposes, as model checking usually relies on a fresh artifact for verification of every new property. This reuse of untanglings yields significant performance gains when querying process model repositories.

3 Preliminaries

This section introduces formalisms that will be used to support subsequent discussions.

3.1 Petri Nets and Net Systems

Petri nets are a well-established formalism for modeling distributed systems, e.g., PAISs. For many high-level process modeling languages, including WS-BPEL and BPMN, there exist mappings to the Petri net formalism [14]. The benefits of such mappings are twofold: (i) rigorous definition of an execution semantics of a high-level language, and (ii) reuse of a mathematical theory of Petri nets for analysis of process models.

This section introduces the basic Petri net terminology and notations.

Definition 3.1 (Petri net)
A *Petri net*, or a *net*, is an ordered triple $N := (P, T, F)$, where P and T are finite disjoint sets of *places* and *transitions*, respectively, and $F \subseteq (P \times T) \cup (T \times P)$ is a *flow* relation. ⌟

A *node* $x \in P \cup T$ is an *input* (an *output*) node of a node $y \in P \cup T$ iff $(x, y) \in F$ ($(y, x) \in F$). By $\bullet x$ ($x\bullet$), $x \in P \cup T$, we denote the *preset* (the *postset*) of x – the set of all input (output) nodes of x. For a set of nodes $X \subseteq P \cup T$, $\bullet X := \bigcup_{x \in X} \bullet x$ and $X\bullet := \bigcup_{x \in X} x\bullet$. A node $x \in P \cup T$ is a *source* (a *sink*) node of N iff $\bullet x = \emptyset$ ($x\bullet = \emptyset$). Given a net $N := (P, T, F)$, by $Min(N)$ ($Max(N)$) we denote the set of all source (all sink) nodes of N. For technical convenience, we require all nets to be T-restricted. A net N is *T-restricted* iff the preset and postset of every transition is non-empty, i.e., $\forall\, t \in T : \bullet t \neq \emptyset \neq t\bullet$.

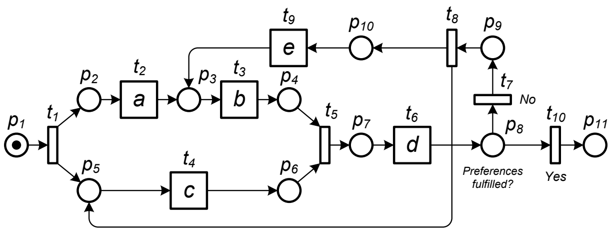


Fig. 2. A net system that captures semantics of the BPMN model in Fig. 1

Often, it is convenient to distinguish between observable and silent transitions of a net; this distinction can be made formal by the means of *labeled* nets.

Definition 3.2 (Labeled net) A *labeled net* is a tuple $N := (P, T, F, \mathcal{T}, \lambda)$, where (P, T, F) is a net, $\mathcal{T}$ is a set of labels, where $\tau \in \mathcal{T}$ is a special label, and $\lambda : T \to \mathcal{T}$ is a function that assigns to each transition in T a *label* in $\mathcal{T}$. ⌟

If $\lambda(t) \neq \tau$, where $t \in T$, then t is *observable*; otherwise, t is *silent*.

Execution semantics of Petri nets is based on states and state transitions and is best perceived as a 'token game'. A state of a net is represented by a *marking*, which describes a distribution of *tokens* on the net's places.

Definition 3.3 (Marking of a net) A *marking*, or a *state*, of a net $N := (P, T, F)$ is a relation $M : P \to \mathbb{N}_0$ that assigns to each place $p \in P$ a number $M(p)$ of *tokens* in p.[1] ⌟

In the sequel, we shall often refer to a marking M as to the multiset containing $M(p)$ copies of place p for every $p \in P$.[2] A *net system* is a Petri net at a certain state/marking.

Definition 3.4 (Net system) A *net system*, or a *system*, is an ordered pair $S := (N, M)$, where N is a net and M is a marking of N. ⌟

In the graphical notation, a common practice is to visualize places as circles, transitions as rectangles, the flow relation as directed edges, and tokens as black dots inside assigned places; see Fig. 2 for an example of a net system visualization.

Whether a transition is *enabled* at a given marking depends on the tokens in its input places. An enabled transition can *occur*, which leads to a new marking of the net.

Definition 3.5 (Semantics of a system) Let $S := (N, M)$, $N := (P, T, F)$, be a system.

- A transition $t \in T$ is *enabled* in S, denoted by $S[t\rangle$, iff every input place of t contains at least one token, i.e., $\forall p \in \bullet t : M(p) > 0$.
- If a transition $t \in T$ is enabled in S, then t can *occur*, which leads to a *step* from S to a system $S' := (N, M')$ via t, where M' is a fresh marking such that $M'(p) := M(p) - \mathbf{1}_F((p,t)) + \mathbf{1}_F((t,p))$, $p \in P$, i.e., t 'consumes' one token from every input place of t and 'produces' one token for every output place of t.[3] ⌟

By $S[t\rangle S'$, we denote the fact that there exists a step from S to S' via t. Note that Fig. 2 shows the labeled net system that formalizes execution semantics of the BPMN model in Fig. 1. Empty rectangles denote silent transitions. Rectangles with labels inside denote observable transitions. These labels refer to the short names shown next to task nodes in Fig. 1. Thus, the full label of transition t_2 in Fig. 2 is "*Get flight preferences*".

A net system induces a set of its occurrence sequences and reachable markings.

[1] $\mathbb{N}_0$ denotes the set of all natural numbers including zero.

[2] We shall write $[p_1, p_1, p_2]$ to denote the marking that puts two tokens at place p_1, one token at place p_2, and no tokens elsewhere.

[3] $\mathbf{1}_F$ denotes the indicator function of F on the set $(P \times T) \cup (T \times P)$.

Definition 3.6 (Occurrence sequence) Let $S_0 := (N, M_0)$ be a net system.

- A sequence of transitions $\sigma := t_1 \dots t_n$, $n \in \mathbb{N}_0$, of N is an *occurrence sequence* in S_0, iff σ is empty or there exists a sequence of net systems $S_1 \dots S_n$ such that for every position i in σ it holds that $S_{i-1}[t_i\rangle S_i$; we say that σ *leads from* S_0 *to* S_n.
- A marking M is *reachable* in S_0, iff $M = M_0$ or there exists an occurrence sequence σ in S_0 that leads from S_0 to (N, M). ⌟

By $\Sigma(S)$ and $[S\rangle$, we denote the set of all occurrence sequences and, respectively, the set of all reachable markings in a net system S. A net system $S := (N, M)$, $N := (P, T, F)$, is *n-bounded*, or *bounded*, iff there exists a number $n \in \mathbb{N}_0$ such that for every reachable marking M' in S and for every place $p \in P$ it holds that the amount of tokens at p is at most n, i.e., $\forall M' \in [S\rangle\, \forall p \in P : M'(p) \leq n$. It is easy to see that the set of all reachable markings in a bounded net system is finite.

3.2 Processes of Net Systems

Occurrence sequences suit well when it comes to describing *orderings* of transition occurrences. In this section, we present *processes* of net systems [15]. One can rely on processes to adequately represent *causality* and *concurrency* relations on transition occurrences. A process of a net system is a net of a particular kind, called *causal net*, together with a mapping from elements of the causal net to elements of the net system.

Definition 3.7 (Causal net) A net $N = (B, E, G)$ is a *causal* net, iff : (i) for each $b \in B$ holds $|\bullet b| \leq 1$ and $|b \bullet| \leq 1$, and (ii) N is acyclic, i.e., G^+ is irreflexive.[4] ⌟

Elements of E are called *events* and elements of B are called *conditions* of N. Two nodes x and y of a causal net $N := (B, E, G)$ are *causal*, iff $(x, y) \in G^+$; otherwise x and y are *concurrent*. A *cut* of a causal net is a maximal (with respect to set inclusion) set of its pairwise concurrent conditions.

One can utilize events of causal nets to represent transition occurrences.

Definition 3.8 (Process) A *process* of a system $S := (N, M)$, $N := (P, T, F)$, is an ordered pair $\pi := (N_\pi, \rho)$, where $N_\pi := (B, E, G)$ is a causal net and $\rho : B \cup E \to P \cup T$ is such that:

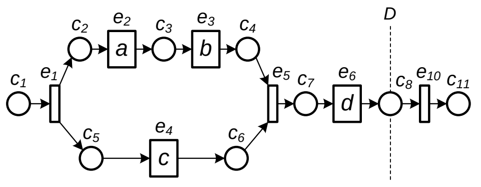
- $\rho(B) \subseteq P$, $\rho(E) \subseteq T$, i.e., ρ preserves the nature of nodes,
- $M = \rho(Min(N_\pi))$, i.e., π starts at M_0, and
- for every event $e \in E$ and for every place $p \in P$ it holds that $|\{(p, t) \in F \mid t = \rho(e)\}| = |\rho^{-1}(p) \cap \bullet e|$ and $|\{(t, p) \in F \mid t = \rho(e)\}| = |\rho^{-1}(p) \cap e \bullet|$,
  i.e., ρ respects the environment of transitions. ⌟

Let $\pi := (N_\pi, \rho)$ be a process of a net system S. It is known that $Min(N_\pi)$ and $Max(N_\pi)$ are cuts [15]. Moreover, every cut of N_π encodes a reachable marking in S.

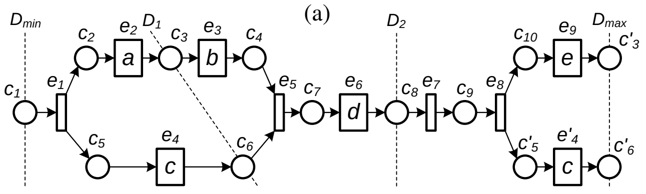
Theorem 3.9 (Cuts and reachable markings, cf. [15, Theorem 3.5])
Let $\pi := (N_\pi, \rho)$, $N_\pi := (B, E, G)$, be a process of a net system S. If $C \subseteq B$ is a cut of N_π, then $M := \rho(C)$ is a reachable marking in S. ⌟

Fig. 3 shows two processes of the net system in Fig. 2. When visualizing processes, conditions $c_i, c'_i \dots$ refer to place p_i, e.g., for the process in Fig. 3(b) it holds that $\rho(c_5) = \rho(c'_5) = p_5$, where p_5 is the place in Fig. 2. Similarly, we employ events $e_i, e'_i \dots$ to denote that they refer to transition t_i, e.g., $\rho(e_4) = \rho(e'_4) = t_4$ for the process in Fig. 3(b). Observe that we distinguish between shapes of events that correspond to silent transitions from those that correspond to observable ones only for clarity considerations.

[4] R^+ denotes the transitive closure of a binary relation R.



(a)



(b)

Fig. 3. Two processes of the net system in Fig. 2

Fig. 3(b) shows a process and four cuts of its causal net N_π. Each cut is defined as a set of conditions that intersect with the respective dashed line. For example, cut D_1 is defined as the set of conditions $\{c_3, c_6\}$. Note that cuts D_{min} and D_{max} are equal to $Min(N_\pi)$ and $Max(N_\pi)$, respectively. Moreover, both cuts D_1 and D_{max} encode the same marking $\rho(D_1) = [p_3, p_6] = \rho(D_{max})$, which is a reachable marking in the net system in Fig. 2, for instance via occurrence sequences $t_1 t_4 t_2$ or $t_1 t_2 t_4 t_3 t_5 t_6 t_7 t_8 t_9 t_4$. Finally, it is easy to see that the set of all processes of the net system in Fig. 2 is *infinite*.

4 Indexing

This section proposes to use untanglings of process models, or more precisely of the corresponding net systems, as data structures that improve the speed of retrieving process instances stored in process model repositories. Similar to database indexes, untanglings require the use of additional storage space to maintain the extra copy of data. However, at this additional cost, they can be used to quickly discover requested process instances without having to iterate over all instances, of which there can be infinitely many.

An *untangling* of a net system is a set of its processes. A process of a system is a static model that describes a *finite* portion of its occurrence sequences, cf. Sect. 3.2. For example, in [16], Jörg Desel suggests to enhance a causal net N_π of a process $\pi := (N_\pi, \rho)$ of a system $S := (N, M)$ with a marking M_π that puts one token at every source condition of N_π and no tokens elsewhere. Then, every occurrence sequence in the fresh system (N_π, M_π) *represents* (via mapping ρ) an occurrence sequence in S. E.g., consider the net system S_π composed of the causal net in Fig. 3(b) and a marking that puts one token at condition c_1 and no tokens elsewhere. Then, occurrence sequence $e_1 e_2 e_3 e_4 e_5 e_6 e_7 e_8 e_9 e'_4$ in S_π represents occurrence sequence $\rho(e_1)\rho(e_2)\rho(e_3)\rho(e_4)\rho(e_5)\rho(e_6)\rho(e_7)\rho(e_8)\rho(e_9)\rho(e'_4)$ $= t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_4$ in the net system in Fig. 2. Observe that in total S_π represents six occurrence sequences of the net system in Fig. 2.

The number of occurrence sequences that are represented in a single process grows combinatorially with the amount of its pairwise concurrent events. This fact makes processes highly suitable for indexing occurrence sequences. Still, it is easy to see that one might often need an *infinite* number of processes to represent — as per the above proposed intuition — all occurrence sequences in a system; e.g., consider the net system in Fig. 2. Clearly, every index must be finite. To this end, we rely on an *enhanced* interpretation of processes, which allows treating a process as a static model that can represent an *infinite* number of occurrence sequences. This enhanced interpretation is formalized in the notion of a *process set system*. In turn, every process set system can be seen as a semantic union of elementary models, called *process systems*.

A *process system* is an abstract model that suggests a way a process of a system can encode a possibly infinite number of occurrence sequences.

Definition 4.1 (Process system) A *process system* of a net system $S := (N, M_0)$ induced by a process π of S is an ordered triple $\mathcal{S}_\pi := (N, M, \pi)$, where M is a marking of N. ⌟

The semantics of process systems – similarly to the semantics of net systems, cf. Definition 3.5 – consists of the transition enablement and transition occurrence rules. The enablement rule of a net system (N, M) depends on the structure of the net N, i.e., on tokens in presets of transitions of the net. Differently, the enablement rule of a process system (N, M, π) relies on the structure of the causal net of π.

Definition 4.2 (Semantics of a process system) Let $\mathcal{S}_\pi := (N, M, \pi)$, $N := (P, T, F)$, $\pi := (N_\pi, \rho)$, $N_\pi := (B, E, G)$, be a process system of a net system S.

- ∘ A transition $t \in T$ is *enabled* in $\mathcal{S}_\pi$, denoted by $\mathcal{S}_\pi[t\rangle$, iff there exists a cut $C \subseteq B$ of N_π and an event $e \in E$ such that $M = \rho(C)$, $\bullet e \subseteq C$, and $t = \rho(e)$.
- ∘ If a transition $t \in T$ is enabled in $\mathcal{S}_\pi$ then t can *occur*, which leads to a step from $\mathcal{S}_\pi$ to $\mathcal{S}'_\pi := (N, M', \pi)$, where M' is a fresh marking such that $(N, M)[t\rangle(N, M')$ holds. ⌟

According to Theorem 3.9, if $C \subseteq B$ is a cut, then $\rho(C)$ is a reachable marking in S. Moreover, if $D \subseteq B$ is a set of conditions, $e \in E$ is an event, and $\bullet e \subseteq D$, then transition $t := \rho(e)$ is enabled in N at the marking $\rho(D)$; this follows immediately from the fact that ρ preserves the nature of nodes and environment of transitions, cf. Definition 3.8. Therefore, a process system $\mathcal{S}_\pi := (N, M, \pi)$, $\pi := (N_\pi, \rho)$, restricts the semantics of the net system (N, M) to those reachable markings that are induced by cuts of N_π and to those transition occurrences that are captured by events of N_π.

Similar to net systems, a sequence of transitions σ is an *occurrence sequence* in a process system $\mathcal{S}_\pi$ if σ is empty or the first transition in σ is enabled in $\mathcal{S}_\pi$ and an occurrence of a transition from σ in $\mathcal{S}$ (except of an occurrence of the last transition in σ) leads to a process system that enables the next transition in σ. We accept that a process π of a net system $S := (N, M)$ *represents* all those occurrence sequences in S which are also occurrence sequences in the process system (N, M, π).

As an example consider a process system $\mathcal{S}_\pi := (N, M, \pi)$, where (N, M) is the net system in Fig. 2 and $\pi := (N_\pi, \rho)$, $N_\pi := (B, E, G)$, is the process in Fig. 3(b). It holds that $\mathcal{S}_\pi$ enables transition t_1. Indeed, there exists cut D_{min} of N_π and event e_1, refer to Fig. 3(b), such that $\rho(D_{min}) = [p_1] = M$, $\{c_1\} = \bullet e_1 \subseteq D_{min} = \{c_1\}$, and $\rho(e_1) = t_1$. An occurrence of t_1 leads to a step from $\mathcal{S}_\pi$ to the process system $(N, [p_2, p_5], \pi)$. It is easy to see that a sequence of transitions $t_1 t_4 t_2 t_3 t_5 t_6$ is an occurrence sequence in $\mathcal{S}_\pi$ which leads to the process system $\mathcal{S}'_\pi := (N, [p_8], \pi)$. Observe that $\mathcal{S}'_\pi$ enables transition t_7 only, whereas the net system $(N, [p_8])$ enables transitions t_7 and t_{10}; recall that N is the net in Fig. 2. There exists only one cut in Fig. 3(b) that induces marking $[p_8]$; this is cut D_2. Finally, it is only event e_7 for which it holds that $\bullet e_7 \subseteq D_2$ and $\rho(e_7) = t_7$. Observe that process system $\mathcal{S}_\pi$ represents *infinitely* many occurrence sequences in the net system in Fig. 2; this is due, for instance, to the fact that the process system $(N, [p_3, p_6], \pi)$ enables transition t_3 via cut D_1. Moreover, $\mathcal{S}_\pi$ represents infinite occurrence sequences; those in which transitions $t_3 \ldots t_9$ can occur infinitely often.

Every process system has its natural boundaries on what portion of process instances it can describe. *Process set systems* aim to overcome these boundaries.

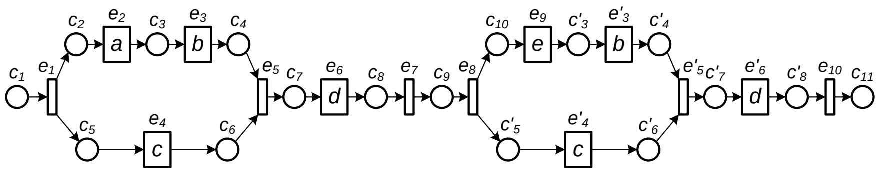


Fig. 4. A process of the net system in Fig. 2

Definition 4.3 (Process set system)
A *process set system* of a net system $S := (N, M_0)$ induced by a set of processes Π of S is an ordered triple $\mathcal{S}_\pi := (N, M, \Pi)$, where M is a marking of N. ⌟

The semantics of a process set system $\mathcal{S} := (N, M, \Pi)$ is 'composed' of all the semantics of individual process systems that are induced by processes in Π.

Definition 4.4 (Semantics of a process set system)
Let $\mathcal{S} := (N, M, \Pi)$, $N := (P, T, F)$, be a process set system.

- A transition $t \in T$ is *enabled* in $\mathcal{S}$, denoted by $\mathcal{S}[t\rangle$, iff there exists a process $\pi \in \Pi$ such that $(N, M, \pi)[t\rangle$ holds.
- If a transition $t \in T$ is enabled in $\mathcal{S}$, then t can *occur*, which leads to a step from $\mathcal{S}$ to $\mathcal{S}' := (N, M', \Pi')$, where M' is a fresh marking such that $(N, M)[t\rangle(N, M')$ holds and $\Pi' := \{\pi \in \Pi \mid (N, M, \pi)[t\rangle\}$. ⌟

As an example consider a process set system $\mathcal{S} := (N, M, \{\pi_1, \pi_2\})$, where (N, M) is, again, the net system in Fig. 2, and π_1 and π_2 are the processes in Figs. 3(a) and 3(b), respectively. The sequence of transitions $t_1 t_2 t_3 t_4 t_5 t_6$ is an occurrence sequence in $\mathcal{S}$ which leads to the process set system $\mathcal{S}' := (N, [p_8], \{\pi_1, \pi_2\})$; again, a sequence of transitions σ is an occurrence sequence in a process set system $\mathcal{S}$ if the first transition in σ is enabled in $\mathcal{S}$ and an occurrence of a transition from σ in $\mathcal{S}$ (except that of the last transition) leads to a process set system that enables the next transition in σ. Transitions t_7 and t_{10} are enabled in $\mathcal{S}'$. Transition t_7 is enabled due to cut D_2 and event e_7 in π_2. Transition t_{10} is enabled due to cut D and event e_{10} in π_1. An occurrence of t_{10} in $\mathcal{S}'$ leads to the process set system $(N, [p_{11}], \{\pi_1\})$, which does not enable any transition.

The process set system $(N, M, \{\pi_1, \pi_2\})$ from the example above represents a big portion of the occurrence sequences in (N, M). Still, it fails to represent all of them. E.g., it does not represent occurrence sequences in which both t_7 and t_{10} occur.

Finally, a *representative* untangling of a net system S is a collection of its processes that induces a process set system which represents *all* the occurrence sequences in S.

Definition 4.5 (Representative untangling) An untangling Π (i.e., a set of processes) of a net system $S := (N, M)$ is *representative* if every occurrence sequence in S is also an occurrence sequence in the process set system (N, M, Π). ⌟

In [17], we demonstrated that: (i) one can always construct a finite representative untangling of a bounded net system, and (ii) a net system S and a process set system $\mathcal{S}$ of S induced by a representative untangling of S are *occurrence net equivalent* [18], i.e., they are two different specifications of the exactly same distributed system.

In [17], we proposed the first algorithm for constructing representative untanglings of bounded net systems. Given the net system in Fig. 2 as input, this algorithm returns two processes shown in Figs. 3(a) and 4 as its representative untangling.

5 Instance-based Retrieval

A representative untangling of a system S is its another specification that represents *all and only* occurrence sequences in S. This section shows how one can employ the unique characteristics of representative untanglings to engineer a process model querying technique. To this end, Sect. 5.1 proposes the *generalized executability* problem and its efficient solution in terms of representative untanglings, whereas Sect. 5.2 uses this solution to formulate basic *query primitives*.

5.1 Executability

Given a net system $S := (N,M)$, $N := (P,T,F)$, and a set of transitions $U \subseteq T$, the *classical executability* problem deals with deciding whether *some* transition in U can ever be 'executed' (can occur) in S. It is a fundamental problem in concurrency theory, e.g., a solution to the executability problem can help deciding *reachability* and *safety* [7].

Definition 5.1 (Executability)
A net system $S := (N,M)$, $N := (P,T,F)$, *can execute some transition* in $U \subseteq T$, iff there exist an occurrence sequence σ in S and a transition $t \in U$ such that t occurs in σ. ⌟

One can solve the executability problem of a system using its representative untangling.

Lemma 5.2 (Executability)
Let Π be a representative untangling of a net system $S := (N,M)$, $N := (P,T,F)$. Then, S can execute some transition in $U \subseteq T$, iff there exist a process $\pi := (N_\pi, \rho)$, $N_\pi := (B,E,G)$, in Π, a transition $t \in U$, and an event $e \in E$ for which it holds that $\rho(e) = t$. ⌟

The proof of Lemma 5.2 is similar to the proof of correctness of a solution to the *generalized executability* problem that is proposed below.

For example, according to Lemma 5.2, one can decide that the net system S in Fig. 2 describes an occurrence sequence that contains transition t_3 using event e_3 of the process π in Fig. 3(a) for which it holds that $\rho(e_3) = t_3$. Moreover, one can use π to generate sample occurrence sequences that contain t_3; these are occurrence sequences in a process system of S induced by π that contain t_3, e.g., $t_1 t_2 t_4 t_3$ is one such sequence.

The executability problem is a *decision problem* on the level of process instances and as such can be naturally applied to formulate queries for searching process models and/or process instances. E.g., a query that relies on a solution to the executability problem can be formulated as follows: "Find all process models that describe a process instance in which a given transition occurs." Alternatively, one can search for exemplary process instances in which a given task occurs. Clearly, one can answer both these questions efficiently using representative untanglings and the result of Lemma 5.2.

In fact, representative untanglings can be used to efficiently solve the *generalized* version of the classical executability problem. As we shall see, this solution broadens the applicability of representative untanglings when searching process model repositories.

Given a net system $S := (N,M)$, $N := (P,T,F)$, and a set of transitions $U \subseteq T$, the *generalized executability* problem deals with deciding whether there exists an occurrence sequence in S which contains *all* the transitions in U.

Definition 5.3 (Generalized executability)
A net system $S := (N,M)$, $N := (P,T,F)$, *can execute all transitions* in $U \subseteq T$, iff there exists an occurrence sequence σ in S such that every transition $t \in U$ occurs in σ. ⌟

The generalized executability problem can be solved using representative untanglings. The proof of correctness of this solution relies on the next corollary.

Corollary 5.4 (Processes and occurrence sequences) Let $\pi := (N_\pi, \rho)$, $N_\pi := (B,E,G)$, be a process of a net system S. Then, there exists an occurrence sequence σ in S such that for every event $e \in E$ it holds that transition $\rho(e)$ occurs in σ. ⌟

Please note that Corollary 5.4 is the special case of Lemma 1 in [16]. Finally, the solution to the generalized executability problem proceeds as follows.

Lemma 5.5 (Generalized executability) *Let Π be a representative untangling of a net system $S := (N,M)$, $N := (P,T,F)$. Then, S can execute all transitions in $U \subseteq T$, iff there exists a process $\pi := (N_\pi,\rho)$, $N_\pi := (B,E,G)$, in Π such that for every transition $t \in U$ there exists an event $e \in E$ for which it holds that $\rho(e) = t$.* ⌟

Proof. We prove each direction of the statement separately.

($\Rightarrow$) *Proof by construction.* Assume that S can execute all transitions in U. According to Definition 5.3, there exists an occurrence sequence σ in S such that every transition $t \in U$ occurs in σ. Then, according to Definition 4.4 and Definition 4.5, there exists a process $\pi := (N_\pi,\rho)$, $N_\pi := (B,E,G)$, in Π such that σ is an occurrence sequence in the process system (N,M,π). Thus, for every transition in σ there exists an event $e \in E$ for which it holds that $\rho(e) = t$. This concludes the proof of the "*only if*" part.

($\Leftarrow$) *Proof by contradiction.* Assume that there exists a process $\pi := (N_\pi,\rho)$, $N_\pi := (B,E,G)$, in Π such that for every transition $t \in U$ there exists an event $e \in E$ of N_π for which it holds that $\rho(e) = t$, but S cannot execute all transitions in U. According to Corollary 5.4, there exists an occurrence sequence σ in S such that for every event $e \in E$ it holds that transition $\rho(e)$ occurs in σ and, hence, every transition $t \in U$ occurs in σ. We reached a contradiction. This concludes the proof of the "*if*" part. ■

For instance, according to Lemma 5.5, one can decide that the net system S in Fig. 2 describes an occurrence sequence that contains transitions t_3, t_7, and t_9 using events e_3, e_7, and e_9 of the process π in Fig. 4 for which it holds that $\rho(e_3) = t_3$, $\rho(e_7) = t_7$, and $\rho(e_9) = t_9$. This conclusion is due to a process system $\mathcal{S}_\pi$ of S induced by π; e.g., $t_1 \dots t_9$ is one of infinitely many occurrence sequences in $\mathcal{S}_\pi$ that contains all the three transitions.

The generalized executability problem can be solved efficiently using untanglings.

Proposition 5.6 *Given a representative untangling Π of a net system $S := (N,M)$, $N := (P,T,F)$, and a set of transitions $U \subseteq T$, the following problem can be solved in the linear time in the size of Π: To decide if S can execute all transitions in U.* ⌟

The proof of Proposition 5.6 is due to Lemma 5.5. Clearly, one can solve the generalized executability problem by visiting each event of the representative untangling once.

Hence, representative untanglings can be used to efficiently retrieve process models and/or exemplary process instances in which all tasks from a given set of tasks occur. Note that, in general, the existence of certain tasks in a process model does not imply the fact that this model describes a process instance in which all these task occur; this is due to conflicting process instances and/or behavioral anomalies, like *deadlocks* [19].

5.2 Query Primitives

This section proposes query primitives that are founded on the definition of the (generalized) executability problem. The seminal construct for all the subsequently proposed primitives is a predicate that given a labeled system S and a set of labels L tests if there exists an occurrence sequence σ in S such that *some* transitions that are labeled with labels in L occur in σ. This seminal predicate can be specialized into four tests:

- `CanOccurOne`(labeled system S, set of labels L) $:= \exists \sigma \in \Sigma(S)\, \exists\, l \in L : l \in \sigma$;[5]
  `CanOccurOne` predicate tests if there exists an occurrence sequence in S which contains at least one transition labeled with some label in L.

[5] $l \in \sigma$ holds if there exists a transition in σ that is labeled with l.

- `CannotOccurOne`(labeled system S, set of labels L) := $\forall\, \sigma \in \Sigma(S)\; \forall\, l \in L : l \notin \sigma$; `CannotOccurOne` predicate tests if there exists no occurrence sequence in S which contains at least one transition labeled with some label in L.
- `CanOccurAll`(labeled system S, set of labels L) := $\exists\, \sigma \in \Sigma(S)\; \forall\, l \in L : l \in \sigma$; `CanOccurAll` predicate tests if there exists an occurrence sequence in S which for every label l in L contains a transition labeled with l.
- `CannotOccurAll`(labeled system S, set of labels L) := $\forall\, \sigma \in \Sigma(S)\; \exists\, l \in L : l \notin \sigma$; `CannotOccurAll` predicate tests if there exists no occurrence sequence in S which for every label l in L contains a transition labeled with l.

For example, one can find all process models that describe a process instance in which task “*Obtain flight price*” occurs by selecting every model M (from a given repertoire of models) for which test `CanOccurOne`(S,{“*Obtain flight price*”}) evaluates to true, where S is a labeled net system that corresponds to M (refer to Sect. 3.1).

Process model repositories often suffer from inconsistent usage of labels, i.e., semantically similar tasks might wear different labels, e.g., “*Get flight quote*” and “*Obtain flight price*”. Consequently, the search procedure that is exemplified above will miss to retrieve the process model in Fig. 1, which can be accepted as a model that is semantically matches the query. To address this issue, we ‘expand’ the predicates. In information retrieval, a *query expansion* is a process of reformulating a *seed query* to improve effectiveness of search results. Every label that is used as input to one of the above proposed seed predicates can be expanded to a set of semantically similar labels, e.g., using the approach in [20]. Accordingly, the predicates get reformulated as follows:

- `CanOccurOneExpanded`(labeled system S, set of sets of labels $\mathcal{L} := \{L_1 \dots L_n\}$) := $\exists\, \sigma \in \Sigma(S)\; \exists\, L \in \mathcal{L}\; \exists\, l \in L : l \in \sigma$; `CanOccurOneExpanded` predicate tests if there exists an occurrence sequence σ in S and a set of labels L in $\mathcal{L}$ such that σ contains a transition labeled with some label in L.
- `CannotOccurOneExpanded`(labeled system S, set of sets of labels $\mathcal{L} := \{L_1 \dots L_n\}$) := $\forall\, \sigma \in \Sigma(S)\; \forall\, L \in \mathcal{L}\; \forall\, l \in L : l \notin \sigma$; `CannotOccurOneExpanded` predicate tests if there exists no occurrence sequence σ in S and no set of labels L in $\mathcal{L}$ such that σ contains a transition labeled with some label in L.
- `CanOccurAllExpanded`(labeled system S, set of sets of labels $\mathcal{L} := \{L_1 \dots L_n\}$) := $\exists\, \sigma \in \Sigma(S)\; \forall\, L \in \mathcal{L}\; \exists\, l \in L : l \in \sigma$; `CanOccurAllExpanded` predicate tests if there exists an occurrence sequence σ in S such that for every set of labels L in $\mathcal{L}$ it holds that σ contains a transition labeled with some label in L.
- `CannotOccurAllExpanded`(labeled system S, set of sets of labels $\mathcal{L} := \{L_1 \dots L_n\}$) := $\forall\, \sigma \in \Sigma(S)\; \exists\, L \in \mathcal{L}\; \forall\, l \in L : l \notin \sigma$; `CannotOccurAllExpanded` predicate tests if there exists no occurrence sequence σ in S such that for every set of labels L in $\mathcal{L}$ it holds that σ contains a transition labeled with some label in L.

For instance, if one is interested in process instances (or models) in which tasks “*Obtain flight price*” and “*Obtain hotel price*” (or semantically similar tasks) occur together, one can start by constructing sets of similar labels, e.g., L_1 :={“*Obtain flight price*”,“*Get flight quote*”} and L_2 :={“*Obtain hotel price*”,“*Get hotel quote*”}. Then, the model in Fig. 1 is a match to the query `CanOccurAllExpanded`(S,$\{L_1,L_2\}$), where S, again, is the net system in Fig. 2. Indeed, the model in Fig. 1 describes process instances in which both tasks “*Get flight quote*” and “*Get hotel quote*” occur. Finally, the model in Fig. 1 can be ranked as one that is less relevant to the query as some other model that is retrieved based on labels “*Obtain flight price*” and “*Obtain hotel price*”, as these labels were initially provided as input, cf. [20] for further details on how results can be ranked.

The proposed predicates explore all possible configurations of the (generalized) executability problem and the above suggested query expansion principle. We provide them for the sake of completeness. However, only *three* (out of the total of eight) checks specify distinct computation patterns. Indeed, every `CannotOccurXY`, `X` $\in$ {'`One`','`All`'}, `Y` $\in$ {'','`Expanded`'}, predicate is the negation of the `CanOccurXY` check. `CanOccurOneExpanded`(S, $\mathcal{L}$) can be implemented as `CanOccurOne`(S, $\bigcup_{L\in\mathcal{L}} L$) test. Note that two out of the three remaining predicates can be expressed in terms of the third one, i.e., `CanOccurOne`(S, L) := $\bigvee_{l\in L}$ `CanOccurAll`(S,$\{l\}$) and `CanOccurAllExpanded` (S, $\{L_1 \ldots L_n\}$):= $\bigvee_{L\in\{\{l_1\ldots l_n\}\,|\,l_1\in L_1,\ldots,l_n\in L_n\}}$ `CanOccurAll`(S,L). However, these two last definitions imply multiple `CanOccurAll` checks which require multiple (and as it turns out unnecessary) traversals of representative untanglings at the computation time.

Because of Proposition 5.6, the `CanOccurAll`(S, L) test can be accomplished in linear time in the size of a representative untangling Π of S; one has to verify if Π contains a process that for each label l in L contains an event that describes an occurrence of a transition which is labeled with l. Similarly, because of Lemma 5.2, when evaluating the `CanOccurOne`(S,L) predicate one needs to search for a process in Π that contains an event that describes an occurrence of a transition labeled with some label in L. Finally, because of Lemma 5.5, in order to fulfill the `CanOccurAllExpanded`(S, $\mathcal{L}$) predicate, there should exist a process in Π that for every set of labels L in $\mathcal{L}$ contains an event which describes occurrence of a transition labeled with some label in L.

6 Evaluation

The proposed querying approach has been implemented and is publicly available as part of the `jBPT` initiative [21].[6] Using this implementation, we conducted an experiment to assess the performance of the approach in terms of querying time and quality of results. The experiment was performed on a computer with a dual core Intel CPU with 2.26 GHz, 4GB of memory, running Windows 7 and SUN JVM 1.7 (with standard allocation of memory). To eliminate load time from the measures, each test was executed six times, and we recorded average times of the second to sixth execution.

The study was conducted on a collection of 448 bounded systems that model processes from financial services, telecommunications, and other domains. These systems were selected from a larger collection of 735 models [19]; systems that do not model concurrency were filtered out as they do not suffer from the state space explosion problem and can be handled efficiently using structural analysis methods.

The study is subdivided into two stages. First, representative untanglings of all systems from the dataset get constructed — the *indexing stage*. Then, constructed untanglings are employed for efficient validation of queries — the *querying stage*.

An extensive experiment that assesses the performance of the indexing stage is reported in [17]. This experiment can be downloaded and reproduced.[7] Next, we summarize basic measures on constructing representative untanglings of the 448 systems. The indexing stage requires 2.72s. Hence, on average, a representative untangling is constructed in 6.06ms; the minimal and maximal construction times are 0.58ms and 221ms, respectively. The average duplication factor, i.e., the average number of times the size of an untangling is larger than the size of its corresponding system (in the number of nodes), is 3.54.

Once constructed, representative untanglings are stored and reused for querying purposes. Table 1 reports average times (in microseconds) of performing `CanOccurOne`

[6] `http://code.google.com/p/jbpt`

[7] `http://code.google.com/p/jbpt/wiki/UntanglingsExperiment`

Table 1. Average times of checking query primitives (in microseconds)

Net systems		`CanOccurOne` (μs)					`CanOccurAll` (μs)				
Size	n	1	2	3	4	5	1	2	3	4	5
1–50	221	1.56	0.93	0.88	0.87	0.84	2.43	1.9	1.9	2.4	2.36
51–100	164	4.27	3.5	3.23	3.06	2.88	7.13	7.17	7.35	7.19	7.39
101–150	44	12.1	10.5	**9.36**	8.87	7.23	22	20.3	20.6	21.7	23.7
151–200	9	25.4	34.8	18.5	16.9	13.6	35	41.2	43.7	46.3	46.2
201–250	7	53.3	49.6	32.7	23	19.8	69	92.6	87.5	100.6	94.2
251–300	3	221.6	147.2	133.8	89.5	81.2	353.9	372	505.9	390.7	424.4
1–300	448	6.35	4.75	4.31	3.72	3.3	10.1	10.3	11.3	11.1	11.4

and `CanOccurAll` checks. The first two columns report on the characteristics of the model collection by providing information on the number ‘n’ of systems within a given ‘Size’ range (measured as the number of nodes). The number of labels used as input to queries ranged from one to five, see the second row and columns three to twelve in the table. Each value is measured as the average time of executing 100 random queries. For example, the value of 9.36 in the fifth row and fifth column in Table 1 reports the average time (in microseconds) of performing `CanOccurOne` checks for the input of three random labels over 44 systems, each of the size within the range from 101 to 150 nodes; in total, 4400 different queries were checked to obtain this average value. The last row in the table shows average times of performing queries over all systems in the collection; these are plotted in Fig. 5(a). One can observe a quasi-linear dependency between the average time of issuing a single check and the size of the set of labels provided as input. The average values for `CanOccurOne` checks show a negative slope. Indeed, as the size of the input set of labels increases, the chance of discovering an occurrence sequence that includes at least one transition labeled with a label from the input set of labels increases as well. On the other hand, more labels in the input sets of `CanOccurAll` queries lead to slower checks as more conditions need to be satisfied.

Table 2 shows average querying times (columns two to seven) and compares quality of retrieved results with label filtering techniques (columns eight to thirteen). The first column lists sizes of input sets of labels; we also vary the sizes of sets used as inputs to `CanOccurAllExpanded` checks. For instance, the value of 7.32 in the fifth row and fifth column in Table 2 is the average time (in milliseconds) of querying 448 systems using the `CanOccurAllExpanded` primitive for an input set that contains two sets, each composed of three labels. Average querying times report on a quasi-linear dependency with the size of the input set of labels, cf. Fig. 5(b).

When searching for process models, one often starts by performing a filtering step, e.g., filtering out those models that do not contain tasks with labels of interest [9,10].

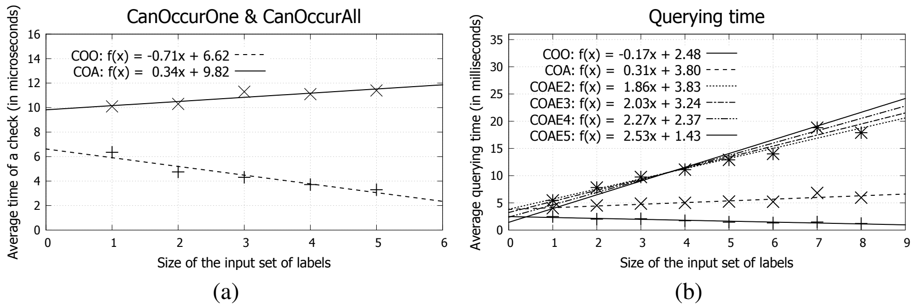


Fig. 5. (a) Average times of performing one check, and (b) average querying times (over 448 systems); `COO` stands for `CanOccurOne`, `COA`—`CanOccurAll`, and `COAE`*n*, $n \in 2..5$, stands for `CanOccurAllExpanded`, where *n* is the size of each set in the input set of sets of labels.

Table 2. Average times of querying a collection of 448 systems (in milliseconds) and average numbers of retrieved systems (using behavioral/structural querying)

Query size	Query time (ms)						Net systems retrieved (using behavioral/structural querying)					
	`COO`	`COA`	`CanOccurAllExpanded`				`COO`	`COA`	`CanOccurAllExpanded`			
			2	3	4	5			2	3	4	5
1	2.48	4.03	5.45	5.05	4.22	3.71	32.5 / 33.6	32.5 / 33.6	67.8 / 70.7	90.2 / 93.5	119.2 / 124	131.8 / 136.5
2	2.02	4.53	7.82	**7.32**	7.05	6.69	68.8 / 72.6	1.23 / 2.84	5.74 / 9.47	15.8 / 21.8	22.4 / 34.8	31.3 / 42.3
3	2.09	4.86	9.75	9.76	9.56	9.28	90.8 / 94.1	0.16 / 0.54	1.4 / 2.64	**2.7 / 6.67**	4.51 / 10.4	9.78 / 19.3
4	1.73	5	11.1	11.2	11.7	11.7	102 / 106.2	0.01 / 0.05	0.13 / 0.62	0.54 / 2.55	2.84 / 5.87	4.13 / 10.2
5	1.46	5.25	12.9	13.4	13.4	13.5	138.6 / 144.5	0 / 0.01	0.02 / 0.16	0.15 / 1	0.67 / 2.95	1.39 / 6.95
6	1.32	5.22	14	14.5	15	15.5	148.8 / 154.4	0 / 0.01	0.02 / 0.13	0.05 / 0.47	0.17 / 1.61	0.55 / 3.7
7	1.48	6.83	18.8	19.2	20.3	21.4	183.5 / 189.6	0 / 0	0 / 0.02	0.02 / 0.32	0.04 / 0.98	0.28 / 2.54
8	1.21	5.96	17.9	18.6	19.4	20.6	194.2 / 200.5	0 / 0	0 / 0.01	0.02 / 0.25	0.05 / 0.67	0.09 / 1.8

(*) `COO` and `COA` stand for `CanOccurOne` and `CanOccurAll`, respectively

Afterwards, computation intensive methods (either structural or behavioral, cf. Section 2) are applied on a much smaller pre-selected collection of models. Query primitives from Section 5.2 can improve effectiveness of existing filtering techniques. To verify this experimentally, we implemented filtering primitives that 'mimic' the primitives from Section 5.2; these fresh primitives analyze process models rather than process instances. For instance, the filtering counterpart of the `CanOccurOne` check from Section 5.2 verifies if a given process model contains a task labeled with some label from a given set of labels. In Table 2, columns eight to thirteen report average numbers of retrieved systems over 100 random queries using both types of primitives. For example, 2.7/6.67 in the sixth row and eleventh column reports that, on average, `CanOccurAllExpanded` primitive which analyzes process instances retrieved 2.7 systems while that which analyzes process models retrieved 6.67 systems. The additional systems selected by analyzing models rather than their instances are the false positives in case one is interested in systems that describe occurrence sequences in which certain transitions occur together.

Finally, we experimented with a querying approach that relies on a model checking technique described in [7]. Model checking of query primitives from Section 5.2 requires, on average, 4ms (based on an implementation that relies on Uma[8]). Though this is approximately 2ms faster than constructing a representative untangling (see above), untanglings can be reused multiple infinite numbers of times when checking suggested query primitives. Observe that the time of 4ms, which is required to perform a single model checking exercise, is comparable with the average time of performing a query over 448 systems that we report in Table 2 (refer to the forth row and third column).[9]

7 Conclusion

This paper proposed a technique for instance-based retrieval of process models from process model repositories. The technique relies on the use of an index data structure which is optimized towards accurate and efficient retrieval of process instances. The use of this index is exemplified via a family of query primitives that are founded on the generalized version of the classical executability problem. The seminal construct for all the primitives is a check on existence of a process instance in which all tasks from a given set of tasks occur. As exemplified, these primitives can be effectively applied in practice, e.g., during process reuse, process compliance, and process standardization

[8] `http://service-technology.org/uma/`

[9] All the experiments reported in this section can be downloaded and reproduced: `http://code.google.com/p/jbpt/wiki/QueryingExperiment`

exercises. Finally, a set of experiments conducted on a large repository of process models from practice showed that during retrieval the use of our index leads to an up to three orders of magnitude speed-up as compared to techniques that rely on model checking.

We envision that our index can be of great use when designing efficient implementations of other query primitives, e.g., those that are founded on relations of *causality* and *concurrency* [22]. Operationalization of these primitives will contribute to maturity of process model query languages, e.g., BPMN-Q [9] and APQL [23].

References

1. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems: Bridging People and Software Through Process Technology. Wiley (2005)
2. Dumas, M., García-Bañuelos, L., Rosa, M.L., Uba, R.: Fast detection of exact clones in business process model repositories. IS **38**(4) (2013) 619–633
3. Jin, T., Wang, J., Rosa, M.L., ter Hofstede, A.H.M., Wen, L.: Efficient querying of large process model repositories. CII **64**(1) (2013) 41–49
4. Awad, A., Sakr, S., Kunze, M., Weske, M.: Design by selection: A reuse-based approach for business process modeling. In: ER. Volume 6998 of LNCS., Springer (2011) 332–345
5. Governatori, G., Sadiq, S.: The Journey to Business Process Compliance. In: Handbook of Research on BPM. IGI Global (2009) 426–454
6. Tregear, R.: Business Process Standardization. In: Handbook on Business Process Management: Part II. Springer Berlin Heidelberg (2010) 307–327
7. Esparza, J., Heljanko, K.: Unfoldings – A Partial-Order Approach to Model Checking. (2008)
8. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying business processes. In: VLDB, ACM (2006) 343–354
9. Awad, A., Sakr, S.: On efficient processing of BPMN-Q queries. CII **63**(9) (2012) 867–881
10. Yan, Z., Dijkman, R.M., Grefen, P.W.P.J.: FNet: An index for advanced business process querying. In: BPM. Volume 7481 of LNCS., Springer (2012) 246–261
11. Dijkman, R.M., Dumas, M., van Dongen, B.F., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. IS **36**(2) (2011) 498–516
12. Jin, T., Wang, J., Wen, L.: Querying business process models based on semantics. In: DASFAA. Volume 6588 of LNCS., Springer (2011) 164–178
13. Kunze, M., Weidlich, M., Weske, M.: Behavioral similarity — A proper metric. In: BPM. Volume 6896 of LNCS., Springer (2011) 166–181
14. Lohmann, N., Verbeek, E., Dijkman, R.M.: Petri net transformations for business processes — A survey. ToPNoC **2** (2009) 46–63
15. Goltz, U., Reisig, W.: The non-sequential behavior of Petri nets. IANDC **57**(2/3) (1983)
16. Desel, J.: Validation of process models by construction of process nets. In: BPM. Volume 1806 of LNCS., Springer (2000) 110–128
17. Polyvyanyy, A., La Rosa, M., Ouyang, C., ter Hofstede, A.H.M.: Untangling concurrent systems for analysis of behavioral properties (2013) `http://eprints.qut.edu.au/56455/`.
18. van Glabbeek, R.J., Vaandrager, F.W.: Petri net models for algebraic theories of concurrency. In: PARLE, Springer (1987)
19. Fahland, D., Favre, C., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Analysis on demand: Instantaneous soundness checking of industrial business process models. DKE (5) (2011)
20. Awad, A., Polyvyanyy, A., Weske, M.: Semantic querying of business process models. In: EDOC, IEEE Computer Society (2008) 85–94
21. Polyvyanyy, A., Weidlich, M.: Towards a compendium of process technologies: The jBPT library for process model analysis. In: CAiSE Forum. Volume 998 of CEUR. (2013)
22. Polyvyanyy, A., Weidlich, M., Conforti, R., La Rosa, M., ter Hofstede, A.H.M.: The 4C spectrum of fundamental behavioral relations for concurrent systems (2013) `http://eprints.qut.edu.au/63443/`.
23. ter Hofstede, A.H.M., Ouyang, C., La Rosa, M., Song, L., Wang, J., Polyvyanyy, A.: APQL: A process-model query language. In: AP-BPM. Volume 159 of LNBIP., Springer (2013)