Graph Matching Algorithms for Business Process Model Similarity Search

Remco Dijkman[1], Marlon Dumas[2], and Luciano García-Bañuelos[2,3]

[1] Eindhoven University of Technology, The Netherlands.
r.m.dijkman@tue.nl
[2] University of Tartu, Estonia.
marlon.dumas@ut.ee
[3] Universidad Autonoma de Tlaxcala, Mexico
lgbanuelos@gmail.com

Abstract. We investigate the problem of ranking all process models in a repository according to their similarity with respect to a given process model. We focus specifically on the application of graph matching algorithms to this similarity search problem. Since the corresponding graph matching problem is NP-complete, we seek to find a compromise between computational complexity and quality of the computed ranking. Using a repository of 100 process models, we evaluate four graph matching algorithms, ranging from a greedy one to a relatively exhaustive one. The results show that the mean average precision obtained by a fast greedy algorithm is close to that obtained with the most exhaustive algorithm.

1 Introduction

As organizations reach higher levels of Business Process Management (BPM) maturity, repositories with hundreds of business process models become increasingly common [18]. For example, the SAP reference model contains over 600 business process models. A similar number of process models can be found in the reference model for Dutch Local Governments [6]. On a larger scale, tool vendors distribute reference model repositories (e.g. the IT Infrastructure Library – ITIL) with over a thousand process models each.[4] These models are used, for example, to document and to communicate internal procedures or to enable the re-design and automation of business processes. In order to effectively fulfil these tasks, tool support is needed to retrieve relevant models from such repositories.

In this paper, we focus on the problem of similarity search in process model repositories: Given a process model or fragment thereof (the *search model*), find those process models in the repository that most closely resemble the search model. The need for similarity search arises in multiple scenarios. For example, when adding a new process model into a repository, similarity search allows one to detect duplication or overlap between the new and the existing process models. Meanwhile, in the context of reference process model repositories, such

[4] See for example CaseWise's ITIL repository (http://www.casewise.com/Gateway/)

as ITIL, similarity search allows one to retrieve reference models that overlap with an existing "as is" process model.

Answering a similarity search query involves determining the degree of similarity between the search model and each model in the repository. In this context, similarity can be defined from several perspectives, including the following.

- Text similarity: based on a comparison of the labels that appear in the process models (task labels, event labels, etc.), using either syntactic or semantic similarity metrics, or a combination of both.
- Structural similarity: based on the topology of the process models seen as graphs, possibly taking into account text similarity as well.
- Behavioural similarity: based on the execution semantics of process models.

In previous work, we evaluated several similarity metrics across all three perspectives [5, 19]. We found that a structural similarity metric based on graph matching achieved the highest retrieval quality (precision and recall). However, the operationalization of this metric is hindered by the fact that the underlying graph matching problem, namely the graph-edit distance problem, is NP-complete [14]. This is not only a theoretical limitation, but a practical one: our experiments show that for real-life process models with more than 20 nodes, exhaustive graph matching algorithms lead to combinatorial explosion. Therefore, heuristics are needed that strike a tradeoff between computational complexity and precision. This paper presents and compares four heuristic algorithms for calculating the similarity of business process models based on graph matching.

The rest of the paper is structured as follows. Section 2 formulates the problem and introduces the structural similarity metric studied in the paper. Section 3 presents four algorithms that provide alternative operationalizations of the structural similarity metric. Section 4 presents an experimental evaluation of these algorithms. Section 5 discusses related work and Section 6 concludes.

2 Preliminaries

This section defines the notion of business process used in this paper and formulates the structural similarity metric used for comparing pairs of process models.

2.1 Business process

A business process is a collection of related tasks that lead to a specified goal. Many modeling notations are available to capture business processes, including Event-driven Process Chains (EPC), UML Activity Diagrams and the Business Process Modeling Notation (BPMN) [20]. In this paper, we seek to abstract as much as possible from the specific notation used to represent process models, to allow for measuring similarity of business processes modeled using different notations. Accordingly, we adopt an abstract view in which a process model is a directed attributed graph, as captured in the following definition.

Definition 1 (Business process graph, Pre-set, Post-set, Source, Sink). *Let $\mathcal{L}$ be a set of labels and $\mathcal{T}$ be a set of types of nodes. A business process graph is a tuple (N, E, τ, λ), in which:*

- *N is the set of nodes;*
- *$E \subseteq N \times N$ is the set of edges; and*
- *$\tau : N \to \mathcal{T}$ is a function that maps nodes to types.*
- *$\lambda : N \to \mathcal{L}$ is a function that maps nodes to labels.*

Let $G = (N, E, \tau, \lambda)$ be a 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. Source nodes are nodes with an empty pre-set and sink nodes are nodes with an empty post-set.

Function τ serves to distinguish between types of nodes. The available types of nodes depend on the notation. In EPCs we can distinguish between at least three types of nodes: functions ('f'), events ('e') and connectors ('c'). Similarly, in BPMN we can distinguish between activities ('a'), events ('e') and gateways ('g'). We could also distinguish between different types of BPMN gateways and events, but it is not the intention of this paper to be exhaustive in this respect.

When abstracting a process model as a process graph, we may drop certain types of nodes. Figure 1 shows two process models (one EPC and one BPMN diagram) and two ways of abstracting them as process graphs. The left column shows the original process models. The middle column shows the corresponding process graphs after the events are abstracted away. Each node is annotated with a pair indicating the node type and the node label. The right column shows the process graphs after events and connectors/gateways are abstracted away. As discussed later, this connector-less abstraction lifts one of the sources of combinatorial explosion when comparing process models using graph matching.

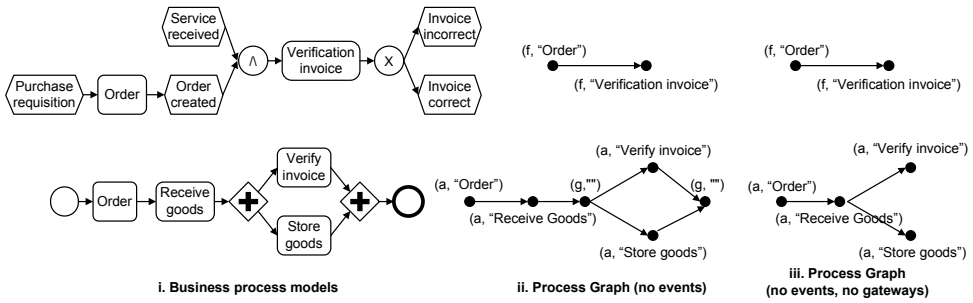

Fig. 1. Two processes and their graphs

2.2 Business process similarity metric

To compare pairs of process graphs, we define a metric based on on the notion of graph edit distance [4]. The graph edit distance between two graphs is

the minimal cost of transforming one graph into the other. Transformations are captured as sequences of elementary transformation operations. Each elementary operation has a cost, which is given by a cost function. Conceptually, a graph-edit distance algorithm must try possible combinations of transformation operations and return the one with the minimal total cost. We consider the following elementary transformation operations.

- Node substitution: a node from one graph is substituted for a node from the other graph.
- Node insertion/deletion: a node is inserted into or deleted from a graph.
- Edge insertion/deletion: an edge is inserted into or deleted from a graph.

We consider cost functions that return a constant value for insertion and deletion of nodes and edges (e.g. a cost of 0.5 for edges and 0.2 for nodes). Meanwhile, we assume that the cost of a node substitution is one minus the similarity of the nodes. The similarity of nodes is determined by the similarity the node labels and types. We introduce a predicate cs ('can substitute') that holds iff one type of node can substitute another type of node (e.g. an EPC function can substitute a BPMN activity). For a given pair of nodes, if cs does not hold, the similarity of these nodes is undefined ($\perp$). If cs holds, their similarity is determined using the string-edit distance of the node labels as defined below.

Definition 2 (String edit distance, Node similarity). *Let s and t be two strings and let $|x|$ be the length of a string x. The string edit distance of s and t, denoted $ed(s,t)$ is the minimal number of atomic string operations needed to transform s into t or vice versa. The atomic string operations are: inserting a character, deleting a character or substituting a character for another.*

Let $G_1 = (N_1, E_1, \tau_1, \lambda_1)$ and $G_2 = (N_2, E_2, \tau_1, \lambda_2)$ be two graphs and $n_1 \in N_1$ and $n_2 \in N_2$ two nodes. The similarity of n_1 and n_2 is:

$$Sim(n_1, n_2) = \begin{cases} 1.0 - \frac{ed(\lambda_1(n_1), \lambda_2(n_2))}{\max(|\lambda_1(n_1)|, |\lambda_2(n_2)|)} & \text{if } \mathrm{cs}(\tau_1(n_1), \tau_2(n_2)) \\ \perp & \text{otherwise} \end{cases}$$

For example, if 'f' and 'a' can substitute each other, then the string edit distance between 'Verify invoice' and 'Verification invoice' from figure 1 is seven; substitute 'y' for 'i' and insert 'cation'. Consequently, the string edit similarity is $1.0 - \frac{7}{20}$. Algorithms for computing the string edit distance are well known [9].

String-edit distance is only one possible similarity metric between labels. In separate work, we studied other label similarity metrics based on word stemming and synonym relations [5]. However, the purpose of the present paper is not to evaluate label similarity metrics, but rather to evaluate algorithms that, given a label similarity metric, compute a similarity measure between pairs of process models. Therefore, the choice of label similarity metric is secondary.

Given the above, we define the graph edit distance as follows.

Definition 3 (Graph edit distance). *Let $G_1 = (N_1, E_1, \tau_1, \lambda_1)$ and $G_2 = (N_2, E_2, \tau_1, \lambda_2)$ be two graphs. Let $M : N_1 \nrightarrow N_2$ be a partial injective mapping*

that maps nodes in G_1 to nodes in G_2. Let $\text{dom}(M) = \{n_1 | (n_1, n_2) \in M\}$ *be the domain of* M *and* $\text{cod}(M) = \{n_2 | (n_1, n_2) \in M\}$ *be the codomain of* M*.*

Given an $n \in N_1 \cup N_2$*,* n *is substituted iff* $n \in \text{dom}(M)$ *or* $n \in \text{cod}(M)$*.* subn *is the set of all substituted nodes. A node* $n_1 \in N_1$ *is deleted from* G_1 *(or inserted in* G_2*) iff it is not substituted. A node that is deleted from* G_2 *(or inserted in* G_1*) is defined similarly.* skipn *is the set of all inserted and deleted nodes.*

Let $(n_1, m_1) \in E_1$ *be an edge in* E_1*.* (n_1, m_1) *is deleted from* G_1 *(or inserted in* G_2*) if and only if there do not exist mappings* $(n_1, n_2) \in M$ *and* $(m_1, m_2) \in M$ *and edge* $(n_2, m_2) \in E_2$*. Edges that are deleted from* G_2 *(or inserted in* G_1*) are defined similarly.* skipe *is the set of all inserted and deleted edges. An edge is substituted if it is not inserted or deleted.*

The graph edit distance that is induced by the mapping M *is:*

$$|\text{skipn}| + |\text{skipe}| + 2 \cdot \Sigma_{(n_1,n_2)\in M}(1 - \text{Sim}(n_1, n_2))$$

The graph edit distance *of the two graphs is the minimal possible distance induced by some mapping.*

For example, given the two process graphs in figure 1.iii, we can create a mapping from 'Order' to 'Order', and from 'Verification invoice' to 'Verify invoice'. The graph edit distance induced by this mapping is: $2.0+4.0+2.0\cdot(0.0+0.35) = 6.7$ (2 inserted nodes, 4 deleted/inserted edges and 2 substituted nodes).

Finally, we define the graph edit similarity metric as follows.

Definition 4 (Graph edit similarity). *Let* $G_1 = (N_1, E_1, \lambda_1)$ *and* $G_2 = (N_2, E_2, \lambda_2)$ *be two graphs. Let* $M : N_1 \nrightarrow N_2$ *be a partial injective mapping that maps nodes in* G_1 *to nodes in* G_2 *and let subn, skipn and skipe be the sets of substituted nodes, inserted or deleted nodes and inserted or deleted edges as defined in definition 3. Furthermore, let* $0 \leq \text{wsubn} \leq 1$*,* $0 \leq \text{wskipn} \leq 1$ *and* $0 \leq \text{wskipe} \leq 1$ *be the weights that we assign to substituted nodes, inserted or deleted nodes and inserted or deleted edges, respectively.*

The fraction of inserted or deleted nodes, denoted fskipn*, the fraction of inserted or deleted edges, denoted* fskipe *and the average distance of substituted nodes, denoted* fsubsn*, are defined as follows.*

$$\text{fskipn} = \frac{|\text{skipn}|}{|N_1|+|N_2|} \quad \text{fskipe} = \frac{|\text{skipe}|}{|E_1|+|E_2|} \quad \text{fsubn} = \frac{2.0\cdot\Sigma_{(n,m)\in M} 1.0-\text{Sim}(n,m)}{|\text{subn}|}$$

The graph edit similarity induced by the mapping M *is:*

$$1.0 - \frac{\text{wskipn} \cdot \text{fskipn} + \text{wskipe} \cdot \text{fskipe} + \text{wsubn} \cdot \text{fsubn}}{\text{wskipn} + \text{wskipe} + \text{wsubn}}$$

The graph edit similarity *of two graphs is the maximal possible similarity induced by a mapping between these graphs.*

For example, using the weights wsubn = 1.0, wskipn = 0.1 and wskipe = 0.3, the graph edit similarity that is induced by the mapping that maps 'Order' to 'Order', and 'Verification invoice' to 'Verify invoice' in figure 1 is: $1.0 - \frac{0.1\cdot0.33+0.3\cdot1.0+1.0\cdot0.7}{0.1+0.3+1.0} \approx 0.73$. This is also the maximal possible similarity induced by a mapping and, hence, this is the graph edit similarity of the two graphs.

3 Algorithms

To compute the graph edit similarity of two process graphs, we must find the mapping that induces the maximal similarity. We could construct all possible mappings and return the one with maximal similarity. However, this approach has factorial complexity. Accordingly, this section presents four possible heuristic algorithms to address this problem.

3.1 Greedy Algorithm

We first propose a greedy algorithm (Algorithm 1) that incrementally constructs a mapping between a pair of process graphs. The algorithm starts by marking all possible pairs of nodes from the two graphs as open pairs. (For all algorithms we assume that pairs of nodes that cannot substitute each other, as defined in definition 2, are not considered.) In each iteration, the algorithm selects an open pair that most increases the similarity induced by the mapping, and adds this pair to the mapping.[5] The selected pair consists of two nodes. Since each node can only be mapped once, the algorithm removes from the set of open pairs, all pairs in which one of the selected nodes appears. The algorithm iterates until there is no open pair left that can increase the similarity induced by the mapping.

The algorithm is in $O(n^3)$ where n is the number of nodes of the largest graph. Indeed, in the first iteration we consider up to n^2 open pairs, in the second iteration $(n-1)^2$ open pairs, etc. And $\Sigma_{i=1}^{n} i = n(n+1)(2n+1)/6$.[6] Also, the algorithm has a quadratic space complexity (the set of open pairs). Unfortunately, the algorithm may lead to a suboptimal mapping, because it selects an open pair that most increases the similarity induced by the mapping at a particular time, but in doing so, it may discard open pairs that would increase the similarity induced by mapping at a later iteration.

For example, in figure 1 the open pair ('Order', 'Order') is chosen in the first iteration, because adding this pair to the mapping increases the similarity score most. All open pairs in which 'Order' appears are then removed from the set of open pairs. In the second iteration, the open pair ('Verification invoice', 'Verify invoice') is chosen. All open pairs in which either 'Verification invoice' or 'Verify invoice' appears are removed. This leaves no open pairs and the algorithm returns the mapping { ('Order', 'Order'), ('Verification invoice', 'Verify invoice')}.

3.2 Exhaustive Algorithm with Pruning

The second algorithm (Algorithm 2) recursively explores all possible mappings, but when the recursion tree reaches a certain size, the algorithm prunes it to keep only the mappings with the highest similarity. In the extreme case, the algorithm is thus exponential, but the pruning parameters will control its complexity.

[5] The similarity induced by a mapping is given by function s as per definition 4.

[6] Computing the graph edit similarity induced by a mapping can be done in constant time (amortized), because when we add a pair we already know the graph edit similarity induced by the existing mapping.

Algorithm 1: Greedy algorithm

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

init
  $\mathsf{openpairs} \Leftarrow N_1 \times N_2$
  $\mathsf{map} \Leftarrow \emptyset$
begin
  while *exists* $(n, m) \in \mathsf{openpairs}$*, such that* $s(\mathsf{map} \cup \{(n, m)\}) > s(\mathsf{map})$ *and there does not exist another pair* $(o, p) \in \mathsf{openpairs}$*, such that* $s(\mathsf{map} \cup \{(o, p)\}) > s(\mathsf{map} \cup \{(n, m)\})$ **do**
    $\mathsf{map} \Leftarrow \mathsf{map} \cup \{(n, m)\}$
    $\mathsf{openpairs} \Leftarrow \{(o, p) \in \mathsf{openpairs} | o \neq n, p \neq m\}$
  end
  return $s(\mathsf{map})$
end

The algorithm starts by initializing the set of unfinished mappings to an empty mapping, with all nodes from the two graphs mapped as 'free' to be mapped. It repeatedly prunes the set of unfinished mappings and performs a step in which finished mappings are added to the set of finished mappings and unfinished mappings are extended with an additional pair of nodes. It repeats this until there are no more unfinished mappings. It then returns the finished mapping with the highest similarity score.

The pruning function (shown separately) tests if the set of unfinished mappings has reached the size 'pruneat' (a parameter of the algorithm). If it has, it returns a set of mappings (of size 'pruneto') with the highest similarity score.

The recursion step is also shown in a separate function. The recursion step takes each unfinished mapping. If the unfinished mapping has no nodes that are free to be mapped, the mapping is added to the set of finished mappings. Otherwise, the algorithm takes each possible combination of pairs of free nodes and creates a new unfinished mapping in which that pair is added to the existing unfinished mapping (and the nodes from the pair are removed from the sets of free nodes). It includes pairs in which free nodes are not mapped (i.e. they are removed from the sets of free nodes, but not added to the unfinished mapping).

For example, in figure 1 the set of unfinished mappings is initialized to $\{(\emptyset, \{O, V\}, \{O, R, V, S\})\}$ (using the first letter of node labels as identifier). In the first step, the algorithm takes this unfinished mapping and, since neither $\{O, V\}$ nor $\{O, R, V, S\}$ is empty, it generates a mapping for each combination of a node from $\{O, V\}$ and a node from $\{O, R, V, S\}$, i.e. $(\{(O, O)\}, \{V\}, \{R, V, S\})$, $(\{(O, R)\}, \{V\}, \{O, V, S\})$, $(\{(O, V)\}, \{V\}, \{O, R, S\})$, ... It also generates one mapping for each possible removal of a node from one of the two sets, generating: $(\emptyset, \{V\}, \{O, R, V, S\})$, $(\emptyset, \{O\}, \{O, R, V, S\})$, $(\emptyset, \{O, V\}, \{R, V, S\})$, ... The generated mappings form the new set of unfinished mappings. In the next step the generation of new unfinished mappings is repeated for each of these mappings.

This example illustrates that the set of unfinished mappings increases exponentially. Pruning will keep the size of the set within acceptable bounds. Suppose

Algorithm 2: Exhaustive algorithm with pruning

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

function prune(unfinished)
begin
  if |unfinished| < pruneat **then**
    return unfinished
  else
    return a set pruned, such that pruned $\subseteq$ unfinished, |pruned| = pruneto and
    $\forall p \in$ pruned : $\neg\exists u \in$ unfinished : $s(\text{first}(u)) > s(\text{first}(p))$
  end
end

function step(unfinished)
begin
  newunfinished $\Leftarrow \emptyset$
  foreach (map, free$_1$, free$_2$) $\in$ unfinished **do**
    if (free$_1 = \emptyset$) $\vee$ (free$_1 = \emptyset$) **then**
      finished $\Leftarrow$ finished $\cup$ map
    else
      newunfinished $\Leftarrow$ newunfinished $\cup$
        $\{(\text{map} \cup \{(f_1, f_2)\}, \text{free}_1 - \{f_1\}, \text{free}_2 - \{f_2\}) | f_1 \in \text{free}_1, f_2 \in \text{free}_2\} \cup$
        $\{(\text{map}, \text{free}_1 - \{f_1\}, \text{free}_2) | f_1 \in \text{free}_1\} \cup$
        $\{(\text{map}, \text{free}_1, \text{free}_2 - \{f_2\}) | f_2 \in \text{free}_2\}$
    end
  end
  return newunfinished
end

init
  unfinished $\Leftarrow \{(\emptyset, N_1, N_2)\}$
  finished $\Leftarrow \emptyset$
begin
  repeat
    unfinished $\Leftarrow$ prune(unfinished)
    unfinished $\Leftarrow$ step(unfinished)
  until unfinished $= \emptyset$
  return s(map), *such that* map $\in$ finished *and* s(map) *is maximal*
end

that 'prune at' is set to 2 and 'prune to' is set to 1, then the set of unfinished mappings will be pruned after the first step, because the set will have reached a size of 2. It will be pruned back to a set the set $\{(\{(O, O)\}, \{V\}, \{R, V, S\})\}$ of size 1, because this mapping has the highest similarity score.

3.3 Process Heuristic Algorithm

The third algorithm is a variation of the exhaustive algorithm. It also builds a recursion tree of possible mappings, but it starts by mapping the source nodes of the business process graphs, then mapping nodes that immediately follow

the source nodes, etc. Since it is plausible that nodes closer to the start of a process should be mapped to nodes closer to the start of the other process (and conversely), this should yield a higher-quality pruning. Indeed, the algorithm is more likely to prune mappings with node pairs that are further apart in terms of their distance to the starts of their processes.

Algorithm 3 shows only the initialization of the algorithm and the 'step' function. The 'prune' function and the algorithm itself are the same as for the exhaustive algorithm 2. The algorithm starts by initializing the set of unfinished mappings to an empty mapping with all nodes marked as 'free' and all source nodes marked as 'current'. With each 'step' the algorithm takes an unfinished mapping. If the unfinished mapping has no 'current' nodes, the mapping is added to the set of finished mappings. Otherwise, the algorithm takes each possible combination of pairs of 'current' nodes and creates a new unfinished mapping in which that pair is added. The nodes from the pair are removed from the sets of free nodes. The current nodes are set to include the post-sets of the nodes from the pairs. Only free nodes are included in the sets of current nodes. Pairs in which 'current' nodes are not mapped are also included. The algorithm assumes that process graphs always have source nodes, an assumption that is valid for common process modeling notations (e.g. EPC, BPMN, BPEL).

Algorithm 3: Process heuristic algorithm

function step(unfinished)
begin
  newunfinished $\Leftarrow \emptyset$
  foreach $(\mathsf{map}, \mathsf{free}_1, \mathsf{free}_2, \mathsf{curr}_1, \mathsf{curr}_2) \in \mathsf{unfinished}$ **do**
    if $(\mathsf{curr}_1 = \emptyset) \vee (\mathsf{curr}_1 = \emptyset)$ **then**
      finished $\Leftarrow$ finished $\cup$ map
    else
      newunfinished $\Leftarrow$ newunfinished $\cup$
        $\{(\mathsf{map} \cup \{(c_1, c_2)\}, \mathsf{free}_1 - \{c_1\}, \mathsf{free}_2 - \{c_2\}, (\mathsf{curr}_1 \cup c_1\bullet) \cap (\mathsf{free}_1 - \{c_1\}), (\mathsf{curr}_2 \cup c_2\bullet) \cap (\mathsf{free}_2 - \{c_2\}))|c_1 \in \mathsf{curr}_1, c_2 \in \mathsf{curr}_2\} \cup$
        $\{(\mathsf{map}, \mathsf{free}_1 - \{c_1\}, \mathsf{free}_2, (\mathsf{curr}_1 \cup c_1\bullet) \cap (\mathsf{free}_1 - \{c_1\}), \mathsf{curr}_2)|c_1 \in \mathsf{curr}_1\} \cup$
        $\{(\mathsf{map}, \mathsf{free}_1, \mathsf{free}_2 - \{c_2\}, \mathsf{curr}_1, (\mathsf{curr}_2 \cup c_2\bullet) \cap (\mathsf{free}_2 - \{c_2\}))|c_2 \in \mathsf{curr}_2\}$
    end
  end
  return newunfinished
end

init
  unfinished $\Leftarrow \{(\emptyset, N_1, N_2, \{n|n \in N_1, \bullet n = \emptyset\}, \{n|n \in N_2, \bullet n = \emptyset\})\}$
  finished $\Leftarrow \emptyset$

For example, in figure 1 the set of unfinished mappings is initialized to $\{(\emptyset, \{O, V\}, \{O, R, V, S\}), \{O\}, \{O\}\}$ (using the first letter of the labels to identify each node). In the first step, the algorithm will take this unfinished mapping and, because neither set of current nodes ($\{O\}$ nor $\{O\}$) is empty. From this map-

ping, it generates one mapping in which the current nodes are mapped, generating $\{(\{(O,O)\},\{V\},\{R,V,S\}),\{V\},\{R\}\}$. It also generates mappings for each possible removal of a current node, generating $\{(\emptyset,\{V\},\{O,R,V,S\}),\{V\},\{O\}\}$ and $\{(\emptyset,\{O,V\},\{R,V,S\}),\{O\},\{R\}\}$. The generated mappings form the new set of unfinished mappings. This example illustrates that the set of unfinished mappings explodes less rapidly for this algorithm than for the exhaustive algorithm. It also illustrates that mappings of nodes closer to the start of the process are explored first.

3.4 A-star Algorithm

The fourth algorithm (Algorithm 4) is based on the well-known A-star heuristic search, which has been applied to the problem of graph matching in [14]. In each step, the algorithm selects the existing partial mapping map with the maximal graph edit similarity. The algorithm then takes a node n_1 from graph G_1 that has not yet been mapped, and creates a mapping between this node and every node n_2 of G_2 such that n_2 does not already appear in map. Let us say that m such nodes n_2 exist. The algorithm then creates m new mappings, by adding (n_1, n_2) to map. In addition, one mapping is created where (n_1, ϵ) is added to map (ϵ is a "dummy" node). This latter pair represents the case where node n_1 has been deleted. This step is repeated until all nodes from G_1 are mapped. It can be proven that the result is an optimal mapping.

The number of steps performed by the algorithm is bounded by $O(n^2m)$ where n and m are the number of nodes in G_1 and G_2. However, $O(m^n)$ partial mappings need to be maintained during the search [14]. To reduce the memory requirements, we modified the algorithm so as to avoid mapping nodes with very different labels. If the string-edit similarity between two node labels is less than a cut-off value, we do not consider the possibility of mapping these nodes.

For example, if we consider the models in figure 1 and a cut-off value of 0.6, two mappings, {('Order', 'Order')} and {('Order', ϵ)}, are created in the first iteration. Since other candidate node pairs have a string-edit similarity smaller than the cut-off value, no mapping is created for them. In the second iteration, the algorithm selects the mapping {('Order', 'Order')} and creates two new mappings {('Order', 'Order'), ('Verification invoice', 'Verify invoice')} and {('Order', 'Order'), ('Verification invoice', ϵ)}. The algorithm stops in the third iteration with a complete mapping {('Order', 'Order'), ('Verification of invoice', 'Verify invoice')} and with nodes 'Receive goods' and 'Store goods' being considered as insertions. Thus, the algorithm discards the two partial mappings {('Order', ϵ)}, {('Order', 'Order'), ('Verification of invoice', ϵ)}.

4 Evaluation

In this section, we present an experimental evaluation of the algorithms discussed above in terms of quality of retrieval results and in terms of execution time.

Algorithm 4: A-star algorithm

input: two business process graphs $G_1 = (N_1, E_1, \lambda_1)$, $G_2 = (N_2, E_2, \lambda_2)$
init
  open $\Leftarrow \{\{(n_1, n_2)\} | n_2 \in N_2 \cup \{\epsilon\}, Sim(n_1, n_2) > \mathsf{ledcutoff} \vee n_2 = \epsilon\}$, for some $n_1 \in N_1$
begin
  while open $\neq \emptyset$ **do**
    select map $\in$ open, such that s(map) is maximal
    open $\Leftarrow$ open $-$ {map}
    if dom(map) $= N_1$ **then**
      return s(map)
    else
      select $n_1 \in N_1$, such that $n_1 \notin$ dom(map)
      foreach $n_2 \in N_2 \cup \{\epsilon\}$, such that either $n_2 \notin$ cod(map) and $Sim(n_1, n_2) >$ ledcutoff or $n_2 = \epsilon$ **do**
        map$'$ $\Leftarrow$ map $\cup \{(n_1, n_2)\}$
        open $\Leftarrow$ open $\cup \{$map$'\}$
      end
    end
  end
end

4.1 Experimental setup

We derived an experimental dataset from the SAP reference model. This is a collection of 604 business process models (described as EPCs) capturing business processes supported by the SAP enterprise system. We randomly extracted 100 business process models from this collection and tagged them as "document models". On average each model contained 21.6 nodes with a minimum of 3 and a maximum 130 nodes. The average size of node labels was 3.8 words. From the 100 document models we randomly extracted 10 models. These models became the "search query models". We modified some of these models to investigate the effect of certain types of changes (for example taking a subgraph) on the performance of the algorithms. We did not observe any noteworthy effects. Therefore, we will only present overall averaged results.

Next, we manually compared each of the 1000 pairs (search model, document model) and ranked their degree of similarity on a 1-7 Likert scale. This manual comparison was done by three process modeling experts, including the first author of this paper. For a given search model sq, we sorted the 100 pairs (sq, document model) in descending order according to the human expert score. Finally, for each algorithm, we applied it to each pair ("search query model", "document model") and sorted the results (for each of the 10 queries) in descending order according to the similarity score retrieved by the algorithm. The resulting sorted lists were used to calculate the average precision.

The algorithms depend on several parameters:

– wskipn, wsubn and wskipe which denote the weight given to node deletion, node substitution and edge deletion (see Definition 4).
– ledcutoff (label edit cut-off): a number between zero and one representing the minimum similarity that two nodes must have so that we can consider their substitution. For example, if the cut-off is 0.5 "Pay Invoice" and "Pay Allowance" will not be mapped since their similarity is 0.3.
– pruneat is the maximum allowed size of the recursion tree. When the recursion tree reaches this level, it is pruned, down to a size of pruneto.

In the experiments, we considered multiple variants of each algorithm corresponding to different parameter settings. The implementation of the proposed algorithms (and several others) can be found in the "Graph Matching Analysis Plug-in" of the ProM process mining and analysis framework.[7]

4.2 Results

Table 1 shows the mean average precision and the average execution times of the similarity search techniques under study. Average precision is a measure commonly used to evaluate the quality of search techniques that return ranked lists of results [3]. It is the average of the precision scores at each point where a relevant document appears in the ranked list. Given a ranked list of results of size n, the average precision is $\Sigma_{j=1}^{n}(precision[j] \times rel[j])/R$, where R is the number of relevant documents, $rel[j]$ is one if the document of rank j in the list is relevant, zero otherwise, and $precision[j] = \Sigma_{k=1}^{j} rel[k]/j$ (i.e. the precision at rank j). Intuitively, average precision is higher when relevant documents appear earlier in the ranked list. The mean average precision of a search technique over a given set of queries is the mean of the average precision of the technique over each of the queries.

As explained above, each algorithm has a number of parameters. The mean average precisions reported in the table correspond to the scores obtained for the best possible settings of each algorithm. All four algorithms depend on parameters wskipn, wsubn and wskipe explained above. We varied each of these parameters from 0 to 1 in increments of 0.1 and ran the experiments with all possible combinations of parameter values in this range. By analyzing the mean average precisions obtained for every combination of parameter values, we noticed that the "Greedy", "Exhaustive" and "Heuristic" algorithms give their best results for settings such that $2 \times (\text{wskipn} + \text{wskipe}) \sim \text{wsubn}$. One can notice that the optimal parameter settings for these three algorithms (Table 1) closely satisfy this condition.

The exhaustive and the process heuristic algorithm rely on parameters pruneat and pruneto to determine when should pruning occur and to what extent. We tested different values of pruneat (50, 100, 200, etc.) and different ratios pruneat/pruneto (0.1, 0.2, etc.). We found that a value of pruneat = 100 is sufficient. Larger values do not improve the results significantly, but they degrade

[7] `http://prom.sourceforge.net`

performance. Similarly we found that a ratio pruneat/pruneto = 0.1 is sufficient, larger ratios do not significantly improve the outcome. Accordingly, we settled for pruneat = 100 and pruneto = 10.

The A-star algorithm relies on a parameter ledcutoff. Again, we experimented with different values of this parameter and found that a value of 0.5 yields optimal results among those that we were able to test. We could not experiment with values significantly below 0.5, because if the threshold is too low, the memory requirements of the A-star algorithm grow substantially and the performance degrades to the point of making the technique impractical. This is the reason why this parameter is important for the A-star algorithm, whereas the other algorithms rely on pruning. A side-effect of using the ledcutoff parameter is that the algorithm favours insertions and deletions over substitutions. To compensate for this effect, the values of wskipn and wskipe need to be set higher than wsubn, in other words, deletions/insertions need to be given higher weight than substitutions. For the A-star algorithm, we noticed that all settings of wskipn, wskipe and wsubn that satisfy this condition given high average precisions.

The A-star algorithm slightly outperforms the others in terms of mean average precision. Looking closer, we noticed that A-star outperforms all other techniques in 6 out of 10 queries and yields equal results in a seventh query. It slightly underperforms the others in queries 6, 8 and 10.

Table 1 also displays the average execution time of 5 runs of each algorithm. For these measurements, we used the parameter settings giving the highest mean average precision. In each run, we executed all 10 queries, i.e. 1000 pairwise process model comparisons in total. All tests were conducted on a laptop with a dual core Intel processor, 2.4 GHz, 4 GB memory, running Mac OSX and SUN Java Virtual Machine version 1.6 (with 512MB of allocated memory).

Table 1. Summary of results

Algorithm	wskipn	wsubn	wskipe	**Mean avg. precision**	**Execution time**
Greedy	0.1	0.9	0.4	0.84	3.8 sec.
Exhaustive	0.1	0.8	0.2	0.82	53.7 sec.
Process Heuristic	0.1	0.8	0.2	0.83	14.2 sec.
A-star	0.2	0.1	0.7	0.86	15.7 sec.

Not surprisingly, the greedy algorithm is considerably faster than all others. Its execution time per search query is less than half a second. The A* and the process heuristic algorithms have comparable execution times – around 1.5 seconds per query. The exhaustive algorithm is significantly slower.

5 Related Work

To the best of our knowledge there exist eight other initiatives that address algorithms for measuring the similarity between business process models or similar models [1, 7, 10–12, 15, 16, 21]. Of these initiatives five present algorithms to measure the similarity between business process models [7, 10–12, 15], two present

algorithms to measure the similarity between state machines [16, 21] and one presents algorithms to measure the similarity between a business process and a set of execution traces [1]. Our algorithms are the only ones that are validated for use in similarity search. Nejati et al. [16] validate their algorithms, but for suitability as a technique for merging state machines. Wombacher [21] validates the correlation of the similarity scores found by his technique with similarity scores assigned according to human judgement. We have done a similar validation in previous work [19]. The different initiatives have very different bases for computing the similarity. Nejati et al. [16] use a combination of label similarity, comparison of the depth of a state-machine fragment in a hierarchical state-machine and bi-similarity of the fragment. Wombacher [21] evaluates three algorithms; one is based on conformance of a set of execution traces (first generated from a process model) to a business process, similar to the work by Van der Aalst et al. [1]; the other two are based on comparison of the language that is represented by a state machine. Li et al. [10] compare process models by 'counting' the number of high-level change operations needed to transform one process into another. This can be seen as a specialized case of edit distance, using a specific set of transformation operations. Like this paper Minor et al. [15] use graph edit distance as a basis for comparing process models. Lu and Sadiq [11] measure the presence or absence of 'features' in proces models as a basis for comparison. Madhusudan et al. [12] use an algorithm known as 'similarity flooding' [13]. Ehrig et al. [7] use a combination of structural properties of process models and similarity of labels of tasks, based on the distance of words in those labels in terms of whether they are, for example, synonyms (which we called 'semantic similarity' in previous work [5]). Table 2 summarizes the related work on business process model comparison.

Table 2. Comparison of related work

Paper	**Similarity of**	**Validated**	**Basis for similarity**
This paper	process models	for similarity search	edit distance
Nejati et al. [16]	state machines	for merging state machines	bi-similarity
Wombacher [21]	state machines	for correlation with human judgement	process conformance language construction
Li et al. [10]	process models	no	change patterns
Minor et al. [15]	process models	no	edit distance
Lu and Sadiq [11]	process models	no	featues
Madhusudan et al. [12]	process models	no	similarity flooding
Van der Aalst et al. [1]	process model and execution traces	no	process conformance
Ehrig et al. [7]	process models	no	semantic similarity
Grigori et al. [8]	service protocols	for similarity search	edit distance (A*)

The algorithms that we studied are based on graph edit distance [4]. However, the actual distance metric we used is different from traditional graph edit distance metrics. Our metric considers the ratio between the actual graph edit

distance and the maximum possible distance. In addition, we added various parameters to the algorithms and fine-tuned these parameters for the computation of similarity of business process models. Of the algorithms that we tested, the greedy algorithm and the exhaustive algorithm with pruning are general algorithms to solve recursive problems. The process heuristic algorithm is similar to Neuhaus and Bunke's planar graph matching algorithm [17]. The main difference is that their algorithm starts with a random pair of graph nodes for comparison, while we assume that business process models have source nodes and sink nodes and we start by mapping source nodes. The A-star algorithm that we present is due to Messmer [14]. We adapted it to exclude mappings of node pairs that are deemed improbable based on the string edit distance of their labels. This algorithm was also applied in [8] for similarity search of service protocol specifications captured in BPEL and WSCL. The authors showed that the algorithm performs well on a small collection of service protocols (5 protocols and variants).

6 Conclusion

Among the four process similarity search techniques presented in this paper, the greedy and the A-star ones offer the most interesting tradeoffs. The A-star algorithm offers a slightly better mean average precision but is significantly slower. Still, the execution times of the A-star algorithm can be acceptable for repositories of a few hundred models. The other two techniques, based on an exhaustive search with pruning, offer a less attractive quality/scalability tradeoff.

The graph matching algorithms studied in this paper attempt to establish 1-to-1 correspondences between nodes in the compared process models (i.e. a node in a process model is related to at most one node in the other process model). One can think of variants of these algorithms that would calculate 1-to-N or N-to-M correspondences, e.g. algorithms that would consider the possibility of a node being split into multiple ones or multiple nodes being merged into one. Such graph matching algorithms have been considered in other application domains [2]. We plan to investigate such variants in future work.

This paper focuses on similarity of business processes with respect to tasks and control-flow relations between tasks. Other aspects of business processes can be considered when determining similarity, e.g. data and resources. Also, process models can be annotated with information that helps to determine the similarity more precisely, such as ontological information [7] and textual documentation. Exploiting such additional information is an avenue for future work.

Acknowledgments . This research was supported by the European Regional Development Fund through the Estonian Centre of Excellence in Computer Science.

References

1. W. van der Aalst, A.K. Alves de Medeiros, and A. Weijters. Process Equivalence: Comparing two process models based on observed behavior. In *Proc. of BPM 2006*, volume 4102 of *LNCS*, pages 129–144. Springer, 2006.

2. R. Ambauen, S. Fischer, and H. Bunke. Graph edit distance with node splitting and merging, and its application to diatom identification. In *Graph Based Representations in Pattern Recognition*, volume 2726 of *LNCS*, pages 259–264. Springer, 2003.
3. C. Buckley and E.M. Voorhees. Evaluating evaluation measure stability. In *Proc. of the ACM SIGIR Conference*, pages 33–40, 2000.
4. H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.
5. R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling. Similarity of business process models: Metrics and evaluation. Working Paper 269, BETA Research School, Eindhoven, The Netherlands, 2009.
6. Documentair structuurplan, Accessed: 20 Feb. 2009. `http://www.model-dsp.nl/`.
7. M. Ehrig, A. Koschmider, and A. Oberweis. Measuring similarity between semantic business process models. In *Proc. of APCCM 2007*, pages 71–80, 2007.
8. D. Grigori, J.C. Corrales, and M. Bouzeghoub. Behavioral matchmaking for service retrieval: Application to conversation protocols. *Inf. Syst.*, 33(7-8):681–698, 2008.
9. I Levenshtein. Binary code capable of correcting deletions, insertions and reversals. *Cybernetics and Control Theory*, 10(8):707–710, 1966.
10. C. Li, M. U. Reichert, and A. Wombacher. On measuring process model similarity based on high-level change operations. Technical Report TR-CTIT-07-89, CTIT, Enschede, The Netherlands, 2007.
11. R. Lu and S. Sadiq. On the discovery of preferred work practice through business process variants. In *Proc. of ER 2007*, volume 4801 of *LNCS*, pages 165–180. Springer, 2007.
12. T. Madhusudan, L. Zhao, and B. Marshall. A case-based reasoning framework for workflow model management. *Data Knowl. Eng.*, 50(1):87–115, 2004.
13. S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. of ICDE 2002*, pages 117–128, 2002.
14. B. Messmer. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs*. PhD thesis, University of Bern, Switzerland, 1995.
15. M. Minor, A. Tartakovski, and R. Bergmann. Representation and structure-based similarity assessment for agile workflows. In *Proc. of the Intl. Conf. on Case-Based Reasoning*, volume 4626 of *LNAI*, pages 224–238. Springer, 2007.
16. S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. Matching and merging of statecharts specifications. In *Proc. of ICSE 2007*, pages 54–63, 2007.
17. M. Neuhaus and H. Bunke. An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification. In *Proc. of the Intl. Workshop on Structural and Syntactic Pattern Recognition*, volume 3138 of *LNCS*, pages 180–189. Springer, 2004.
18. M. Rosemann. Potential pitfalls of process modeling: part a. *Business Process Management Journal*, 12(2):249–254, 2006.
19. B. F. van Dongen, R. M. Dijkman, and J. Mendling. Measuring similarity between business process models. In *Proc. of CAiSE 2008*, volume 5074 of *LNCS*, pages 450–464. Springer, 2008.
20. M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, Berlin, Germany, 2007.
21. A. Wombacher. Evaluation of technical measures for workflow similarity based on a pilot study. In *Proc. of CoopIS 2006*, volume 4275 of *LNCS*, pages 255–272. Springer, 2006.