Aligning Business Process Models

Remco Dijkman*, Marlon Dumas†, Luciano García-Bañuelos‡ and Reina Käärik†
*Eindhoven University of Technology, The Netherlands
Email: R.M.Dijkman@tue.nl
†University of Tartu, Estonia
Email: marlon.dumas@ut.ee, reinak@math.ut.ee
‡Universidad Autónoma de Tlaxcala, Mexico
Email: lgbanuelos@gmail.com

***Abstract*—This paper studies the following problem: given a pair of business process models, determine which elements in one model are related to which elements in the other model. This problem arises in the context of merging different versions or variants of a business process model or when comparing business process models in order to display their similarities and differences. The paper investigates two approaches to this alignment problem: one based purely on lexical matching of pairs of elements and another based on error-correcting graph matching. Using a set of models taken from real-life scenarios, the paper empirically shows that graph matching techniques yield a significantly higher precision than pure lexical matching, while achieving comparable recall.**

***Keywords*-business process model; model comparison; model merging;**

I. Introduction

As any other type of model, business process models are expected to undergo changes, for example as a result of the adoption of new work practices, or as a result of company re-structurings or mergers, or because of revisions aimed at correcting or refining an existing business process model. In such cases, business process modelers need to compare different versions or variants of a business process model in order to understand and to approve changes.

In this paper, we focus on one aspect of this problem, namely *business process model alignment* [1].[1]. Given a pair of process models, the goal is to automatically establish a relation between the elements of one model (e.g. tasks or events) and the elements in the other model.

Importantly, we assume that there is no *change log* available that would allow one to obtain direct knowledge about the way one model is derived from the other model or from a common model. Instead, we study the problem in the general case, where the models being aligned may have been developed independently or may have been manipulated using different tools, so that a change log is not available. This scenario arises for example in the context of company mergers (e.g. mergers of insurance companies) where business processes designed independently need to be consolidated into a common business process.

Figure 1 shows an example of two related business process models. The processes are represented in the Protos notation. The model contains two types of elements: trigger elements, which represent that something happens, and activity elements, which represent that something is done. The elements are related by means of control-flow relations, which capture the ordering of activities. These two process models are adapted from the set of models that we used to validate our proposal. The left model shows a reference process that concerns applying for a refund for school transportation costs with a municipality. The right model shows an implementation of that process model at a municipality. The relations between the activities from the two process models, which were established by human experts, are represented with solid lines. If multiple activities from one process model are related with one (or more) activities from the other process, a box is drawn around those activities. Establishing the relations between the reference process and the implemented process gives an indication of the level of compliance of the implemented process. Also, if an information system is developed based on the reference process, the relations give an indication of the effort necessary to adapt (to) the information system. However, the figure also illustrates that (automatically) determining the relations between the activities from the two process models is far from trivial.

Contemporary business process modeling tools, such as ARIS or Oracle BPA, support simple forms of process alignment. For example, in Oracle BPA one can export a BPMN process model as an executable process specification in BPEL. Subsequently, this executable BPEL specification can be modifed by a developer. The modified specification can then be transformed back to BPMN and compared to the original BPMN process model. For this comparison, Oracle BPEL relies on unique identifiers attached to process model elements. In contrast, we study the problem of model alignment without assuming the existence of unique identifiers – an assumption that can be made only when the models being compared are derived from one another using the same tool.

[1]This operation is also called "matching", for example in the field of schema matching [20].

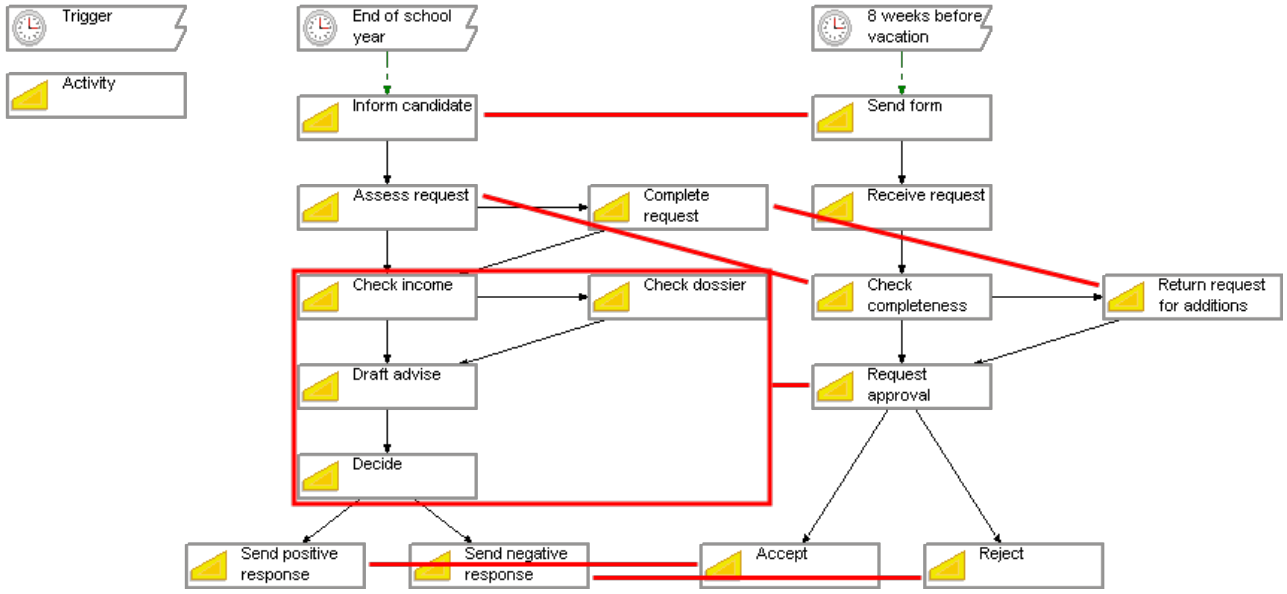

Figure 1. Two business process models with their relations.

The process alignment problem can be addressed from at least three angles [17], [3], [5]:

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

In this paper, we focus on the first two approaches. Specifically, the paper presents two families of techniques: one based purely on lexical matching, and the second based on error-correcting graph matching (i.e. structural matching). We comparatively evaluate these two approaches based on pairs of process models collected from two real-world projects. The results show that structural similarity leads to better results (in terms of precision) than pure lexical similarity.

The paper is structured as follows. Section II introduces some preliminary definitions. Sections III presents the matching techniques considered in this study. Section IV describes the experimental setup and presents the main results. Section V discusses related work and Section VI concludes.

II. Background

A typical business process model is a graph consisting of at least two types of nodes: task nodes and control nodes. Task nodes describe units of work that may be performed by humans or software applications. Control nodes capture the flow of execution between tasks. Process models may also include nodes representing data objects (inputs or outputs of tasks) as well as data mappings to extract and combine data objects. Additionally, process models may include elements to capture resources involved in the performance of tasks.

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) [23]. In this paper, we seek to abstract as much as possible from the specific notation used to represent process models, to allow for matching tasks 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): Let $\mathcal{L}$ be a set of labels. A business process graph is a tuple (N, E, λ), in which:

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

In the techniques we make extensive use of similarity of the node labels to determine the mapping of (task) nodes.

To determine the similarity of node labels, we use the notion of string edit distance.

Definition 2 (String edit distance, String edit similarity): Let s and t be two strings and let $|x|$ represent the number of characters in a string x. The string edit distance of s and t, denoted $\text{ed}(s,t)$ 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. The string edit similarity of s and t, denoted $\text{Sim}(s,t)$ is:

$$\text{Sim}(s,t) = 1.0 - \frac{\text{ed}(s,t)}{\max(|s|,|t|)}$$

For example, the string edit distance between 'Complete request' and 'Receive request' from figure 1 is seven; delete 'Co', substitute 'mpl' with 'Rec', insert 'i' and substitute 't' with 'v'. Consequently, the string edit similarity is $1.0 - \frac{7}{16}$. Techniques for computing the string edit distance are well known [10].

We note that other measures of similarity could be used to compare node labels, including similarity measures based either on lexical databases such as Wordnet.[2] or on semantic annotations attached to tasks. However, as a starting point, we focus on syntactic similarity measures for comparing node labels. Semantic similarity comparison techniques are however not always applicable. For example, in the datasets we consider in our study, most process models are in Dutch, for which there are no comprehensive and free lexical databases available. Also, semantic annotations may or may not be available depending on the source of the process models and the availability of a reference ontology.

III. Matching techniques

In this section we discuss one lexical matching technique, named N-to-M label matching, and two structural matching techniques, named greedy graph matching and A-star graph matching.

A. N-to-M Label Matching

We first consider a straightforward lexical technique to align process models, which we call *N-to-M label matching*. This straightforward technique is included in the study in order to provide a baseline with respect to which other techniques can be benchmarked.

The idea of the technique is to construct a mapping between two process models by matching any node in one model with any node in the other model so long as the similarity between their labels is above a given cut-off value. Here, we could use a number of similarity measures to compare pairs of nodes. For illustration purposes, we use the string-edit similarity presented in Definition 2.

[2]http://wordnet.princeton.edu/

Let us consider the working example (Figure 1). If we use the string-edit similarity, and we set the matching cut-off to 0.5, then the following pairs of nodes are included in the computed mapping: { (Assess request, Receive request), (Complete request, Receive request) }. All other pairs give a string-edit similarity below the cut-off.

In our study, we also considered a variant of string-edit similarity in which we pre-processed each node label using the following steps:

1) Tokenize the node label in order to extract a list of words. In doing so, special symbols and separators (e.g. carriage returns) are discarded.
2) Put all characters in lower-case.
3) Remove so-called 'stop words', which are frequently occurring words, such as "a", "an" and "for".
4) Stem the words using Porter's stemming algorithm [19]. Stemming reduces words to their stem form. For example, "stemming", "stemmed" and "stemmer" all become "stem".
5) Convert back the resulting list of words into a string to make it possible to use the string edit distance for comparison.

These pre-processing steps are intended to eliminate minor variations between labels, so that such minor variations do not have an effect on the construction of the mapping. Having done this pre-processing, we then apply string-edit distance to compare pairs of nodes.

We use the term *N-to-M label matching with stemming* to refer to the variant of the N-to-M label matching in which we apply the above pre-processing steps to each node label before constructing the mapping.

B. Greedy Graph Matching

Structural matching techniques establish a mapping that has an optimal matching score. They compute that score both based on the similarity of the labels of matched nodes, and based on similarity of the presence or absence of edges between matched nodes. Edges can be included in the score by giving a 'bonus' if an edge from one graph can be matched to an edge from the other graph. The structural matching score is strongly related to the graph edit distance [2], but is computed differently as explained below.

We perform two steps to compute the matching score, given two graphs and a mapping between their nodes. First we count the number of:

- Node substitutions: a node in one graph is substituted for a node in the other graph if and only if they are matched.
- Node insertions/deletions: a node is inserted into or deleted from one of the graphs if and only if it is not matched.
- Edge substitution: an edge from node a to node b in one graph is substituted for an edge in the other graph

if and only if node a is matched to node a', node b is matched to node b' and there exists an edge from node a' to node b'.
- Edge insertions/deletions: an edge is inserted into or deleted from one of the graphs if and only if it is not substituted.

Second, we return the weighted average of the fraction of inserted/deleted nodes, the fraction of inserted/deleted edges and the average string edit similarity of the labels of substituted nodes. More precisely, we define the matching score as follows.

Definition 3 (Matching score): 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 matches nodes in G_1 to nodes in G_2, with domain $\mathrm{dom}(M) = \{n|(n,m) \in M\}$ and codomain $\mathrm{cod}(M) = \{m|(n,m) \in M\}$, and let $0 \leq \mathrm{wsubn} \leq 1$, $0 \leq \mathrm{wskipn} \leq 1$ and $0 \leq \mathrm{wskipe} \leq 1$ be the weights that we assign to substituted nodes, inserted or deleted nodes and inserted or deleted edges, respectively.

The set of substituted nodes, denoted $subn$, inserted of deleted nodes, denoted $skipn$, substituted edges, denoted $sube$, and inserted or deleted edges, denoted $skipe$, are defined as follows:

$$
\begin{aligned}
&\mathrm{subn} = \mathrm{dom}(M) \cup \mathrm{cod}(M)\\
&\mathrm{skipn} = (N_1 \cup N_2) - \mathrm{subn}\\
&\mathrm{sube} =\\
&\quad \{(a,b) \in E_1 | (a,a') \in M, (b,b') \in M, (a',b') \in E_2\} \cup\\
&\quad \{(a,b) \in E_2 | (a,a') \in M, (b,b') \in M, (a',b') \in E_1\}\\
&\mathrm{skipe} = (E_1 \cup E_2) - \mathrm{sube}
\end{aligned}
$$

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.

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

The matching score is defined as:

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

For example, given the graphs from figure 1 and the mapping indicated in that figure, we can count the following substitutions, insertions and deletions: substitution of 'Inform candidate' with 'Send form', insertion or deletion of 'Receive request', substitution of 'Assess request' with 'Check completeness', insertion of deletion of the edge between 'Inform candidate' and 'Assess request', insertion of deletion of the edges between 'Send form' and 'Receive request' and between 'Receive request' and 'Check completeness', etceteras. In total there are 9 node substitutions, 1 node insertion/deletion, 5 edge substitutions and 7 edge insertions/deletions.

The formulae above can be used to compute the matching score *for a given mapping*. To compute the optimal matching score, we must try all possible mappings and return the one with the highest score. However, the number of possible mappings is $2^{|N_1|+|N_2|}$. It has been shown the problem of finding an optimal mapping in this setting is an NP-complete problem [15]. To avoid the problems associated with NP-completeness, we use a greedy algorithm to compute the optimal mapping.

The greedy algorithm works as follows. It starts with an empty mapping. It then adds the pair of nodes to the mapping that most increases the graph matching score. If there are multiple such pairs, one is selected randomly. The algorithm continues adding pairs of nodes until there are no more pairs that increase the graph matching score.

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. Also, the algorithm is non-deterministic because it may need to randomly select a pair. However, we show in the evaluation that, in spite of these drawbacks, it produces good results.

C. A-star Graph Matching

An alternative way to tackle structural matching is to perform a search in the mapping space using the well-known A-star algorithm. The idea is as follows. First of all, we start with an empty mapping (with zero matches). In each step, the algorithm selects the existing partial mapping map with the lowest graph edit distance. (If there are multiple such partial mappings, one is chosen randomly.) The algorithm then takes a node n_1 from graph G_1 that has not yet been matched, and creates a match 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 match 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 matched. It can be proven that the result is an optimal mapping, in contrast to the greedy algorithm presented above.

A major issue with the A-star algorithm is that it needs to maintain a large number of partial mappings during the search – $O(m^n)$ in the worst case [15]. Generally speaking, the number of partial mappings that need to be maintained depends on the number of pairs of nodes that can be paired up (i.e. matched). Hence, one way of controlling the space complexity is by only matching "promising" pairs of nodes, as opposed to trying to match all pairs of nodes.

Accordingly, we use the cut-off value introduced in the N-to-M label matching technique, and only allow a pair of nodes to be matched if the string edit similarity is above this cut-off. As discussed later in the paper, this allows us to maintain the memory footprint within practical bounds.

In its basic form, the A-star graph matching technique matches one node in a graph to one node in the other graph. Indeed, in each iteration, the algorithm takes one node that has not yet been matched and matches it with all nodes in the other graph. Once a node has been matched, it is no longer considered. This is different from the greedy algorithm which adds one pair of nodes to the mapping in each iteration, even if this pair contains a node that has already been matched.

We could extend the A-star matching technique in order to consider the possibility of matching a single node in one graph to a group of nodes in the other graph (splitting), or vice-versa (merging), or the possibility of matching a group of nodes in one graph to a group of nodes in the other graph (N-to-M match). This basically means that in each iteration of the algorithm, instead of only considering pairs of nodes, we would consider pairs of groups of nodes. This would lead to a larger number of partial mappings being created at each step. Indeed, in addition to considering the possibility of adding $(n1, n2)$ or $(n1, n3)$ into the current best mapping, we would also need to consider the possibility of adding $(n1, \{n2, n3\})$. When the amount of nodes grows, this leads to a combinatorial explosion.[3]

After experimenting with the possibility of matching groups of nodes during the execution of the A-star algorithm, we realized that this approach was not practical. Accordingly, we opted for an alternative approach in which we let the A-star algorithm run in its basic form in order to produce an initial 1-to-1 mapping – i.e. a mapping where a node in one graph is matched to at most one node in the other graph. Once this initial 1-to-1 mapping is constructed, we apply a post-processing procedure as follows. First, we tag nodes that are not matched as "deleted", in the sense that they appear in one graph but not in the other. We select such a "deleted" node, and we consider the possibility of combining it with an adjacent node that already appears in the mapping. After considering all possible combinations of a "deleted" node with a matched node, we select the combination that would improve the current mapping the most. Next, we update the mapping to reflect this "merging" operation, and we also update the process model in order to combine the two merged nodes into a single one. We repeat this step, until there are no more "deleted" nodes that can be merged with an adjacent node in such a way that this merger would improve the current mapping. Normally, the resulting mapping contains a number of N-to-M matches (N nodes in one graph matched with M nodes in the other graph), in addition to 1-to-1 matches.

[3]The idea of extending the A-star graph matching algorithm with node merging and splitting was also suggested in [3] but the authors did not implement the idea.

IV. Evaluation

We evaluated the performance of the techniques by comparing the matches that they find to matches that process modeling experts found. Section IV-A explains the setup of the experiments and section IV-B presents the main results.

A. Experimental setup

We evaluated the techniques by having them calculate mappings between 17 pairs of process models and comparing the resulting mappings with those found by human experts. These process models were originally modeled in the Protos notation (a proprietary notation supported by the tool Protos commercialized by Pallas Athena). The models were obtained in two projects in the local government domain. Both projects involved comparing processes of a particular municipality to the corresponding reference processes from the 'Documentair StructuurPlan' [7]. On average a model contained 31.8 nodes with a minimum of 9 and a maximum of 81 nodes in a model. The average number of arcs pointing into or out of a node was 1.17. Nodes had labels with on average 2.8 words in a label.

Mappings between pairs of process models were first constructed manually by process experts and then validated with domain experts. In this way, a total of 440 node matches were identified across the 17 pairs of models – i.e. around 26 node matches per pair of process models. On average each model contained 29 nodes.

We used the mappings identified manually to evaluate the performance of the techniques from the previous section, by calculating the precision, recall and F-score of each technique. The precision is the fraction of node matches that is found that is correct (i.e. that are also considered node matches by humans). The recall is the fraction of correct node matches that is found. The F-Score combines precision and recall in one value. More precisely, we define these measures in the context of this paper as follows.

C_t = Matches retrieved by technique t
C_m = Matches identified manually
$precision = |C_t \cap C_m|/|C_t|$
$recall = |C_t \cap C_m|/|C_m|$
$F\text{-}score = 2 * (precision * recall)/(precision + recall)$

The techniques depend on the following parameters:

- wskipn, wsubn and wskipe which denote the weight given to node deletion, node substitution and edge deletion when calculating the graph-edit distance of a mapping (see Definition 3).
- $\mathrm{ledcutoff}$ (label edit cut-off): a number between zero and one representing the minimum similarity that the labels of two activities must have so that we can consider one to substitute the other. If the similarity is smaller than this cut-off value, the techniques will never

Table I
OVERALL RESULTS

Technique	Precision	Recall	F-Score
Lexical N-to-M without stemming	0.78	0.60	0.68
Lexical N-to-M with stemming	0.72	0.60	0.66
A-star	0.83	0.56	0.67
A-star with postprocessing	0.81	0.60	0.69
Greedy	0.89	0.60	0.72

consider that one of the nodes is substituted by the other. For example, if the cut-off is 0.5, two functions labelled "Pay Invoice" and "Pay Allowance" will not be mapped since their similarity is 0.3 which is below the threshold.

The structural matching techniques (Greedy and A-star graph matching) depend on both parameters, the lexical matching technique (N-to-M label matching) only depends on the $\mathrm{ledcutoff}$ parameter.

In the experiments, we considered multiple variants of each technique corresponding to different parameter settings. The implementation of the proposed techniques (and several others) can be found in the "Graph Matching Analysis Plug-in" of the ProM process mining and analysis framework.[4]

B. Experimental results

Table I summarizes the precision, recall and F-score for the techniques from the previous section.

The lexical matching technique only depends on the string-edit similarity cut-off parameter. Tables II and III contain the detailed results obtained with lexical matching for different values of the cut-off parameter. Naturally, in the case of low thresholds, the precision is low, meaning too many false matches are found. In the case of higher thresholds, the precision increases up to 0.8, but at this point the recall falls to around 0.5, meaning too many true matches are *not* found. It appears that the technique reaches its maximum F-score when the cut-off is relatively high (0.75). Above this point, the recall degrades.

Surprisingly, the results show that the stemming procedure does not help to improve the quality of the results. In fact, the precision obtained with stemming is slightly lower than without stemming, although the difference is not significant. This probably comes from the type of dataset: the models in the dataset were developed using the same naming conventions, so that the stemming procedure would have little effect. For example, if a verb appears in past participle in one model, it would also appear in past participle in the other model too. Stemming might be useful when models are developed by completely independent analysts using different conventions, or when no naming conventions are followed.

[4]Available at: http://prom.sourceforge.net.

Table II
RESULTS: N-TO-M LABEL MATCHING WITHOUT STEMMING

Threshold	Precision	Recall	F-Score
0	0.04	1	0.07
0.125	0.04	0.97	0.08
0.25	0.08	0.82	0.15
0.375	0.28	0.7	0.4
0.5	0.45	0.68	0.54
0.625	0.59	0.66	0.62
0.75	**0.78**	**0.6**	**0.68**
0.875	0.81	0.57	0.67
1	0.76	0.52	0.62

Table III
RESULTS: N-TO-M LABEL MATCHING WITH STEMMING

Threshold	Precision	Recall	F-Score
0	0.04	1	0.07
0.125	0.05	0.98	0.09
0.25	0.12	0.83	0.22
0.375	0.29	0.77	0.42
0.5	0.43	0.75	0.55
0.625	0.58	0.65	0.61
0.75	**0.72**	**0.6**	**0.66**
0.875	0.8	0.57	0.66
1	0.74	0.56	0.64

The results of the A-star technique are comparable to those obtained by the N-to-M label matching technique. As expected, the best results are produced by the A-star technique with post-processing. This technique achieves a precision of 0.81, meaning that 19% of the matches found by the technique are false, and a recall of 0.6, meaning that 60% of all matches (identified by humans) are discovered by the technique. The results of the A-star technique depend on the settings of all parameters. Since the A-star technique tries out an exponential number of possible mappings, it is infeasible for use with a string edit similarity cut-off below 0.5. For a cut-off larger than 0.7 the F-score of the technique rapidly decreases.

The greedy technique produces the best results, while at the same time not suffering from the performance bottlenecks of the A-star technique. The greedy technique led to a recall that is similar to the best results that are produced by the other techniques, but with a higher precision (up to 89% precision). It is somehow unexpected that the greedy technique yields better performance than the A-star technique, because a greedy technique can lead to sub-optimal mappings, while the A-star technique is more exhaustive and therefore should produce a better mapping. A plausible explanation is that in the greedy technique, the possibility mapping a group of nodes in one model to a single node in the other model, is considered during the matching, while

the A-star technique only considers such groupings during the post-processing step. As a result, the greedy technique is able to detect earlier that a group of nodes should be mapped to a single one, and it can then proceed to construct the rest of the mapping on this basis.

The greedy technique produced the best results for the settings: wskipn = 0.5, wsubn = 0.8, wskipe = 0.0 and ledcutoff = 0.5.

V. Related Work

Previous research in the area of model-driven software engineering has addressed the problem of comparing and merging models of system behavior, in particular state machines [17]. State machines are much simpler than business process models captured in mainstream process modeling languages such as BPMN, EPC, YAWL. [17] also consider the case of state machines that are hierarchically structured (hierarichal state machines), however, they do not consider constructs such as parallel branching as found for example in UML statecharts. Also, their techniques are mainly based on behavioural matching, while the techniques we study are based on lexical and structural matching.

Applications of graph-matching techniques to process model comparison are documented in [3], [16]. However, the purpose of this work is not to identify relations between elements of business process models. Instead, the goal is to compare one process model to a collection of other process models to determine which one is most similar. This work is related to other work in the area of similarity measures for business process models [21], [11], [24], [12], [13], [8]. This body of work, while related to this paper, does not address the problem of computing correspondences between elements in two process models. [5] presents a more detailed survey of the work in this area.

Techniques also exist for determining differences, rather than similarities, between (different versions of) business process models [4], [6], [9], [22]. These techniques can also be used to determine the differences between business process elements. However, they assume that a change log is available [22] or that mappings between business process elements have already been analyzed [4], [6], [9]. On the basis of these mappings, the latter techniques essentially aim at re-constructing a change log composed of a set of high-level operations, such as "tasks T is moved", "task Y is exchanged with Z", etc. Therefore, the output of the techniques studied in this paper can be given as input to the techniques developed in [4], [6], [9].

The problem of process model alignment can be related to that of data model or schema matching, which has been widely studied in the literature [20]. However, there are important differences between process models and data models. For example, data models and schemas tend to have labelled edges (associations) in addition to labelled nodes. Also, the types of nodes and the attributed attached to nodes are different in these two types of process models (e.g. no control nodes in data models). At first glance these differences would seem trivial. But in practice they lead to techniques for schema matching not being directly applicable to process model alignment. During our experiments, we implemented a graph matching technique originally designed for schema matching, namely Similarity Flooding [14]. After adapting the technique to deal with process models, we tested it on the dataset discussed in this paper. However, the similarity flooding technique led to a poor score (under 0.5 even after fine-tuning). Worst, in the first iteration of the technique, we started with a mapping calculated using the string-edit distance between nodes, that gave an F-score of 0.6. We then iterated several times over this initial match, as described in [14]. But instead of obtaining better mappings, we actually obtained worst mappings with subsequent iterations. The reason probably lies in the fact that edges in process models generally lack labels, while schema matching techniques asssume that both entities (nodes) and associations (edges) are labelled.

Another work close to ours is that by Brockmans et al [1]. In this work, the authors consider different approaches to aligning business process models captured as Petri nets, with an emphasis on linguistic similarity. They consider both syntactic similarity based on string-edit distance, as well as semantic similarity based on ontologies. However, they do not evaluate the proposed techniques experimentally as we do in our paper. Our experiments suggest that approaches based on graph matching are superior to those based purely on linguistic comparison.

VI. Conclusion and Future Research Directions

This paper presents three techniques (in a total of five variants) that are able to automatically match similar tasks from different processes. The techniques are implemented in the ProM process mining and analysis framework as the 'graph matching analysis plug-in'.

We evaluated each technique by comparing the mappings computed by the technique to mappings determined by human experts in a total of seventeen pairs of business processes from two projects. The validation shows that the technique that produces the best results is a greedy graph matching technique that established a mapping based on the topological structure of business processes and similarity of task labels. This technique has a precision of 89%, meaning that 89% of the task mappings that it identifies are correct (i.e.: are also found by humans), and a recall of 60%, meaning that it finds 60% of all correct task mappings.

We claim that this makes the technique usable as an aid to match similar tasks. In particular because the user can rely on the technique to return correct matches (with 89% accuracy). The tool is only suitable as an aid, because the user needs to verify the matches that the technique returns

and needs to manually identify the 40% of the matches that the technique does not detect.

The validation also shows that the greedy graph matching technique, which takes both the topological structure of business processes and task labels into account, produces better results than the lexical techniques, which only take task labels into account. The greedy graph matching technique has an improved precision with respect to the best lexical technique, which has a precision of 72%. The recall is the same for the best lexical technique and the greedy graph matching technique.

Improvement of the techniques is possible, in particular with respect to their recall. The results of the techniques give a strong indication that the recall can be improved by improving the techniques for allowing a single task to be involved in a mapping more than once (also described as 'grouping' in this paper). This is apparent from the fact that extending techniques with grouping only leads to small improvements with respect to their recall (from 59% to 60% for lexical mapping and from 56% to 60% for A-star mapping). However, the total percentage of mappings that can be added when allowing grouping is 23%, suggesting that a much larger improvement with respect to recall is possible. This is a direction for future work. In particular, we plan to consider other graph matching techniques in which grouping of activities could be done more effectively. One possibility is to adapt a technique for graph matching of planar graphs proposed by Neuhaus and Bunke [18], in which a mapping is constructured starting from a node and then exploring the neighbourhood of that node, at which point one could consider grouping activities together. Process models are generally planar graphs (particularly structured process models) and this property can be exploited to achieve better tradeoffs.

Another avenue for future work is to incorporate semantic similarity measures on task labels, using word distance in an ontology or in a dictionary based on, for example, synonyms. The word distance can both be computed for task labels and for the brief descriptions that are often attached to tasks in practice. In this paper, we concentrated on syntactic similarity measures, based on string-edit distance, to compare pairs of node labels. This choice was motivated by the fact that the collection of models used in the experiments were in Dutch, and lexical databases with semantic relationships are not freely available in Dutch (unlike, for example, the Wordnet database in English).

ACKNOWLEDGEMENTS

This research is partly funded by the Estonian Science Foundation and the European Regional Development Fund through the Estonian Centre of Excellence in Computer Science.

REFERENCES

[1] S. Brockmans, M. Ehrig, A. Koschmider, A. Oberweis, and R. Studer, "Semantic alignment of business processes," in *International Conference on Enterprise Information Systems (ICEIS) – Volume 3*, Paphos, Cyprus, May 2006, pp. 191–196.

[2] H. Bunke, "On a relation between graph edit distance and maximum common subgraph," *Pattern Recognition Letters*, vol. 18, no. 8, pp. 689–694, 1997.

[3] J. Corrales, D. Grigori, and M. Bouzeghoub, "BPEL processes matchmaking for service discovery," in *Proceedings of the OTM International Conferences*. Montpellier, France: Springer, November 2006, pp. 237–254.

[4] R. Dijkman, "A Classification of Differences between Similar Business Processes," in *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC'07)*. Annapolis, Maryland, USA: IEEE, 2007, pp. 37–50.

[5] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling, "Similarity of business process models: Metrics and evaluation," BETA Research School, Eindhoven, The Netherlands, Working Paper 269, 2009.

[6] R. M. Dijkman, "Diagnosing differences between business process models," in *BPM*, ser. Lecture Notes in Computer Science, M. Dumas, M. Reichert, and M.-C. Shan, Eds., vol. 5240. Springer, 2008, pp. 261–277.

[7] "Documentair structuurplan," Accessed: 20 February 2009, web page: http://www.model-dsp.nl/.

[8] M. Ehrig, A. Koschmider, and A. Oberweis, "Measuring similarity between semantic business process models," in *Conceptual Modelling 2007, Proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM 2007)*, J. Roddick and A. Hinze, Eds., vol. 67. Ballarat, Victoria, Australia: Australian Computer Science Communications, 2007, pp. 71–80.

[9] J. Küster, C. Gerth, A. Förster, and G. Engels, "Detecting and resolving process model differences in the absence of a change log," in *6th International Conference on Business Process Management (BPM)*. Milan, Italy: Springer, 2008, pp. 244–260.

[10] I. Levenshtein, "Binary code capable of correcting deletions, insertions and reversals," *Cybernetics and Control Theory*, vol. 10, no. 8, pp. 707–710, 1966.

[11] C. Li, M. U. Reichert, and A. Wombacher, "On measuring process model similarity based on high-level change operations," http://eprints.eemcs.utwente.nl/11574/, Centre for Telematics and Information Technology, University of Twente, Enschede, Technical Report TR-CTIT-07-89, December 2007.

[12] R. Lu and S. Sadiq, "On the discovery of preferred work practice through business process variants," in *Conceptual Modeling - ER 2007*, ser. LNCS, vol. 4801. Heidelberg, Germany: Springer, 2007, pp. 165–180.

[13] T. Madhusudan, L. Zhao, and B. Marshall, "A case-based reasoning framework for workflow model management," *Data Knowledge Engineering*, vol. 50, no. 1, pp. 87–115, 2004.

[14] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity flooding: A versatile graph matching algorithm and its application to schema matching," in *International Conference on Data Engineering*. Los Alamitos, CA, USA: IEEE Computer Society, 2002, pp. 117–128.

[15] B. Messmer, "Efficient graph matching algorithms for preprocessed model graphs," Ph.D. dissertation, University of Bern, Switzerland, 1995.

[16] M. Minor, A. Tartakovski, and R. Bergmann, "Representation and structure-based similarity assessment for agile workflows," in *7th International Conference on Case-Based Reasoning*, ser. LNAI, R. Weber and M. Richter, Eds., vol. 4626. Heidelberg, Germany: Springer, 2007, pp. 224–238.

[17] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave, "Matching and merging of statecharts specifications," in *29th International Conference on Software Engineering (ICSE 2007)*, 2007, pp. 54–63.

[18] 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*, ser. LNCS, vol. 3138. Springer, 2004, pp. 180–189.

[19] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[20] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[21] B. F. van Dongen, R. M. Dijkman, and J. Mendling, "Measuring similarity between business process models," in *Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE)*, ser. LNCS, vol. 5074. Springer, 2008, pp. 450–464.

[22] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features - enhancing flexibility in process-aware information systems," *Data Knowl. Eng.*, vol. 66, no. 3, pp. 438–466, 2008.

[23] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, 2007.

[24] A. Wombacher, "Evaluation of technical measures for workflow similarity based on a pilot study," in *Proc. 14th Int'l Conf. on Cooperative Information Systems (CoopIS'06)*, ser. LNCS, vol. 4275. Berlin: Springer, 2006, pp. 255–272.