Improving process model retrieval by accounting for gateway nodes: an ongoing work

S. Montani (1), G. Leonardi (1,2), S. Quaglini (2), A. Baudi (2)

(1) DISIT, Computer Science Institute, Università del Piemonte Orientale, Alessandria, Italy
(2) Dipartimento di Informatica e Sistemistica, Università di Pavia, Italy

Abstract. Process model comparison and similar processes retrieval is a key issue to be addressed in many real world situations, and a particularly relevant one in some applications (e.g., in medicine), where similarity quantification can be exploited in a quality assessment perspective.
In a recent work, we have introduced a framework which allows to: (i) extract the actual process model from the available process execution traces, through process mining techniques; and (ii) compare (mined) process models, by relying on a novel distance measure. The tool has been successfully tested in stroke management.
In this paper, we are further extending that contribution, by explicitly taking into account complex control flow information, represented in the process model as gateway nodes (i.e., AND/XOR join/splits). The work is still ongoing, but, once completed and tested, it will represent a significant advance with respect to the state of the art research in the field, with a range of applications which is of course not limited only to medicine.

1 Introduction

Process model comparison and similar processes retrieval is a key issue to be addressed in many real world situations. For example, when two companies are merged, process engineers need to compare processes originating from the two companies, in order to analyze their possible overlaps, and to identify areas for consolidation. Moreover, large companies build over time huge process model repositories, which serve as a knowledge base for their ongoing process management/enhancement efforts. Before adding a new process model to the repository, process engineers have to check that a similar model does not already exist, in order to prevent duplication.

Particularly interesting is the case of medical process model comparison, where similarity quantification can be exploited in a quality assessment perspective. Indeed, the process model actually implemented at a given healthcare organization can be compared to the existing reference clinical guideline, e.g., to check conformance, or to understand the level of adaptation to local constraints that may have been required. As a matter of fact, the existence of local resource constraints may lead to differences between the models implemented at different hospitals, even when referring to the treatment of the same disease (and to the

same guideline). A quantification of these differences (and maybe a ranking of the hospitals derived from it) can be exploited for several purposes, like , e.g., legal purposes, performance evaluation and funding distribution.

The actual medical process models are not always explicitly available at the healthcare organization. However, a database of process execution traces (also called the "event log") is typically maintained, since all the hospital activities are normally recorded by means of the workflow technology. In this case, process mining techniques [3] can be exploited, to extract process related information (e.g., process models) from log data.

Stemming from these considerations, we have recently started to work at a framework [10], which allows to:

1. extract the actual process model from the available process execution traces, through process mining techniques;
2. perform process model comparison, to fulfill the objectives described above.

Issue 2 has required the introduction of proper metrics, in order to quantify process model similarity. We could rely on an extensive literature when studying this topic (see, e.g., [11, 2]). In particular, since process mining extracts the process model in the form of a graph, where nodes represent activities, and edges represent the control flow, our work is located in the research stream on structural similarity, and on graph-edit-distance-based approaches [2, 4].

The state of the art on structural similarity on process models is represented by the work by Dijkman et al. [4]. In our previous contribution [10], we have extended the work in [4], by: (i) exploiting domain knowledge; (ii) exploiting process mining outputs (e.g., edge reliability, see section 3.1). The use of domain knowledge is particularly significant in medical applications, the field in which our work has been tested so far.

However, as in [4], in [10] we have simplified the mined output, by removing all information about parallel or mutually exclusive execution of activities.

Addressing that limitation is the goal of the present work. In this paper, we will thus describe:

- a proper mapping of the mined output to a graph that includes gateway (i.e., AND/XOR join/splits) nodes;
- an enhanced version of the metric in [10], able to take into account gateway nodes.

This approach will be carefully analysed and experimentally tested in the near future. While the first experiments are foreseen in the field of stroke management (as we did in [10]), the work is meant to be general enough to be exploited in very different applications as well.

The paper is organized as follows: section 2 introduces the process mining technique we are using; section 3.1 summarizes the metrics we defined in [10]; section 3.2 illustrates the enhancements we are now developing, and represents the core contribution of this work. Finally, section 4 addresses conclusions and future work directions.

2 Process Mining and the ProM tool

Process mining describes a family of a-posteriori analysis techniques exploiting the information recorded in event logs, to extract process related information (e.g., process models). Typically, these approaches assume that it is possible to sequentially record events such that each event refers to an activity (i.e., a well defined step in the process) and is related to a particular process instance. Furthermore, some mining techniques use additional information such as the timestamp of the event, or data elements recorded with the event.

Traditionally, process mining has been focusing on discovery, i.e., deriving process models and execution properties from event logs. It is important to mention that there is no a-priori model, but, based on logs, some model, e.g., a Petri net, is constructed. However, process mining is not limited to process models (i.e., control flow), and recent process mining techniques are more and more focusing on other perspectives, e.g., the organisational perspective, the performance perspective or the data perspective. Moreover, as well stated in [6], process mining also supports conformance analysis and process enhancement. In this paper, however, we only deal with the process perspective.

In our work, we are relying on the Process Mining tool called ProM, extensively described in [12]. ProM is a platform independent open source framework which supports a wide variety of process mining and data mining techniques, and can be extended by adding new functionalities in the form of plug-ins.

In particular, we have chosen ProM's *heuristic miner* for mining the process models. Heuristic miner takes in input the event log, and considers the order of the events within every single process instance execution. The time stamp of an activity is used to calculate this ordering. Heuristics miner is tolerant to noise, and can be used to express the main behavior (i.e., not all details) registered in a log. Indeed, some abstract information, such as the presence of composite tasks (i.e., tasks semantically related to their constituent activities by means of the "part-of" relation), cannot be derived by Heuristic Miner, that will only build a model including ground (i.e., not further decomposable) activities. On the other hand, it can mine the presence of short distance and long distance dependencies (i.e., direct or indirect sequence of activities), and information about parallelism, with a certain reliability degree (see also section 3.1). The output of the mining process is provided as a graph, also called "dependency graph", where nodes represent activities, and arcs represent control flow information.

3 Extending graph edit distance for process model comparison

3.1 Exploiting domain knowledge and process mining output

In order to compare process models, in [10] we have introduced a distance definition that extends previous literature contributions [4, 2] by properly considering domain knowledge, and additional information, learned through process mining.

In particular, since mined process models are represented in the form of graphs (where nodes represent activities and edges provide information about the control flow), we define a distance based on the notion of graph edit distance [2]. Such a notion calculates the minimal cost of transforming one graph into another by applying insertions/deletions and substitutions of nodes, and insertions/deletions of edges.

While string edit distance looks for an *alignment* that minimizes the cost of transforming one string into another by means of edit operations, in graph edit distance we have to look for a *mapping*. A mapping is a function that matches nodes to nodes, and edges to edges. Among all possible mappings, we will select the one that leads to the minimal cost, having properly quantified the cost of every type of edit operation.

As in [4], we have provided a normalized version of the approach in [2].

Moreover, with respect to both [2] and [4], we have introduced two novel contributions:

1. we calculate the cost of node substitution $fsubn$ (see Definition 4 below) by applying **taxonomic distance** [9, 8] (see Definition 1), and not string edit distance on node names as in [4]. Indeed, we have organized the various activities executable in our domain in a taxonomy, where activities of the same type (e.g., Computer Assisted Tomography (CAT) *with* or *without* contrast) are connected as close relatives. The use of this definition allows us to explicitly take into account this form of domain knowledge, since the distance between two activities is set to the normalized number of arcs on the path between the two activities themselves in the taxonomy (see Definition 1);
2. we add a cost contribution related to edge substitution ($fsube$ in Definition 4 below), that incorporates information learned through process mining, namely (i) the percentage of traces that have followed a given edge, and (ii) the reliability of a given edge, i.e., of the control flow relationship between two activities. Both items (i) and (ii) are outputs of heuristic miner (see Definitions 2 and 3 below).

Formally, the following definitions apply:

Definition 1: Taxonomic Distance.

Let α and β be two activities in the taxonomy t, and let γ be the closest common ancestor of α and β. The *Taxonomic Distance* $dt(\alpha, \beta)$ between α and β is defined as:

$$dt(\alpha, \beta) = \frac{N_1 + N_2}{N_1 + N_2 + 2 * N_3}$$

where N_1 is the number of arcs in the path from α and γ in t, N_2 is the number of arcs in the path from β and γ, and N_3 is the number of arcs in the path from the taxonomy root and γ.

Definition 2: Reliability. The reliability of the edge ei assessing that activity a directly follows activity b in sequence (i.e., ei is an arc from b to a) is calculated as [13]:

$$rel(ei) = \frac{|a > b| - |b > a|}{|a > b| + |b > a| + 1}$$

where $|a > b|$ is the number of occurrences in which activity a directly follows activity b in the event log, and $|b > a|$ is the number of occurrences in which activity b directly follows activity a.
A negative reliability value means that we must conclude that the opposite pattern holds, i.e., activity b follows activity a. Indeed, the reliability of a relationship (e.g., activity a follows activity b) is not only influenced by the number of occurrences of this pattern in the logs, but is also (negatively) determined by the number of occurrences of the opposite pattern (b follows a). However, edges with a negative reliability will not appear in the dependency graph (due to threshold mechanisms and proper heuristics [13], that rule them out). Therefore, we will deal with reliability values $\in (0, 1)$
Definition 3: Percentage of traces. The percentage of traces that crossed edge ei, assessing that activity a directly follows activity b in sequence, is calculated as:

$$ptrace(ei) = \frac{|a > b|_t}{|ALLTRACE|}$$

where $|a > b|_t$ is the number of traces in which activity a directly follows activity b in the event log, and $|ALLTRACE|$ is the total number of available traces in the event log. With this definition, the percentage of traces $\in [0, 1]$.
Definition 4: Extended Graph Edit Distance. Let $G1 = (N1, E1)$ and $G2 = (N2, E2)$ be two graphs, where Ei and Ni represent the sets of edges and nodes of graph Gi. Let M be a partial injective mapping [4] that maps nodes in $N1$ to nodes in $N2$ and let $subn$, $sube$, $skipn$ and $skipe$ be the sets of substituted nodes, substituted edges, inserted or deleted nodes and inserted or deleted edges with respect to M. In particular, a substituted edge connects a pair of substituted nodes in M. 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 $fsubn$, are defined as follows:

$$fskipn = \frac{|skipn|}{|N1| + |N2|}$$

$$fskipe = \frac{|skipe|}{|E1| + |E2|}$$

$$fsubn = \frac{2 * \sum_{n,m \in M} dt(n,m)}{|subn|}$$

where n and m are two mapped nodes in M.

The average distance of substituted edges $fsube$ is defined as follows:

$$fsube = \frac{\sum_{(n1,n2),(m1,m2) \in M} (|rel(e1) - rel(e2)| + |ptrace(e1) - ptrace(e2)|)}{|sube|}$$

where edge $e1$ (connecting node $n1$ to node $m1$) and edge $e2$ (connecting node $n2$ to node $m2$) are two substituted edges in M, $rel(ei)$ is the reliability of edge ei, and $ptrace(ei)$ is the percentage of traces that crossed edge ei.

The extended graph edit distance induced by the mapping M is:

$$ext_{edit} = \frac{wskipn * fskipn + wskipe * fskipe + wsubn * fsubn + wsube * fsube}{wskipn + wskipe + wsubn + wsube}$$

where $wsubn$, $wsube$, $wskipn$ and $wskipe$ are proper weights $\in [0, 1]$.

The extended graph edit distance of two graphs is the minimal possible distance induced by a mapping between these graphs. To find the mapping that leads to the minimal distance we resort to the greedy algorithm described in [4].

3.2 Exploiting gateway nodes comparison

The dependency graph may contain complex control flow information (i.e., other than sequence) between the mined process activities. Actually, parallelism and mutual exclusion can be identified during the mining process. Activity nodes in the dependency graph, in this case, can contain up to three types of information (see "action B" in figure 1, left part): an input label (e.g., XOR, AND, or more complex combinations, such as an AND of two XORs); the current activity name; an output label. We will focus on simple labels (i.e. AND or XOR), for the sake of clarity.

The input label provides information about the incoming flow. Two situations may take place:

1. if only one edge enters the node (see node B in figure 1), the input label means that the current node is preceded by a split. E.g., it has to be carried out in mutual exclusion with one or more other activities, if the label is a XOR;
2. if, on the other hand, multiple edges enter the node, the input label means that the current node is preceded by a join (see node D in figure 1).

The output label works dually: multiple outgoing edges mean that the node is followed by a split (see node A in figure 1), a single outgoing edge means that the node is followed by a join (see node B in figure 1).

Such a representation can make the dependency graph very complex, and difficult to read. Moreover, a direct application of the metric in section 3.1 to this graph is not possible. Actually, the information in input and output labels was simply disregarded in the metric in section 3.1 (in line with the approach in [4]).

In the current version of our work, on the other hand, we are exploiting this information, by mapping the dependency graph onto another graph structure, in which gateway nodes (AND/XOR join/splits) are explicitly introduced, and control flow patterns are clearer. The new graph structure is also more similar to classic workflow representation languages [1].

Specifically, somehow similarly to [5], the mapping works as follows:

- if a node has only one incoming edge, we remove the input label, if any;
- otherwise, if the node has multiple incoming edges, we introduce a proper join gateway node, as the input label indicates (i.e., AND or XOR); connecting arcs are also introduced;
- if a node has only one outgoing edge, we remove the output label, if any;
- otherwise, if the node has multiple outgoing edges, we introduce a proper split gateway node; connecting arcs are also introduced.

For instance, figure 1 shows the mapping of a small dependency graph (on the left) into a graph containing both activity and gateway nodes (on the right).

While the metric in section 3.1 properly compares activity nodes, a new contribution has to be added to it, in order to account for gateway nodes.

To this end, we proceed as follows:

1. if the two gateway nodes x and y are of different types (i.e. a XOR and an AND) their distance is set to 1;
2. if the two gateway nodes x and y are of the same type (e.g., two ANDs), we have to calculate the difference between:
   (a) the incoming gateway nodes;
   (b) the incoming activity nodes;
   (c) the outgoing gateway nodes;
   (d) the outgoing activity nodes.

As regards item 2.(b), let S1 be the sequence of incoming activity nodes of the first gateway node x; let S2 be the sequence for the second gateway node y. Without loss of generality, suppose that S2 is not longer than S1. In order to compare S1 ad S2, we try all possible permutations in the order of the activity nodes in S2, and take the one that leads to the minimal distance with respect to S1. The distance between the two sequences is the average of the distance between single elements (i.e., pairs of activities), over the length of the longest sequence. The distance between a pair of activities is calculated resorting to taxonomic distance (see Definition 1 in section 3.1). Every activity in S2 that cannot be mapped to any activity in S1 (because S1 is longer than S2) contributes with a distance of 1.

Item 2.(d) works analogously.

Items 2.(a) and 2.(c) are simpler: identical incoming (respectively, outgoing) gateway nodes in the two sequences (e.g., two AND) provide a contribution of 0; different gateway nodes (i.e., an AND and a XOR) provide a contribution of 1. As above, we then calculate the average over the length of the longest sequence of gateway nodes. This procedure is obviously a simplification, since incoming gateway nodes may have other gateway nodes in input as well, but we do not consider this (recursive) information. Similar considerations hold for outgoing gateway nodes. This choice was motivated by computational complexity issues, but could be reconsidered in the future.

The four contribution are then combined in a weighted average $df(x, y)$, in which weights can be experimentally set.

In the current version of our framework, the $fsubn$ contribution of Definition 4 (see section 3.1) is therefore modified as follows:

$$fsubn = \frac{2 * (\sum_{n,m \in M_A} dt(n, m) + \sum_{x,y \in M_G} df(x, y))}{|subn|}$$

where M_A represents the set of mapped activity nodes in the mapping M, M_G represents the set of mapped gateway nodes in M, and $df(x, y)$ calculates the distance between two gateway nodes x and y in M_G by applying the procedure described above.

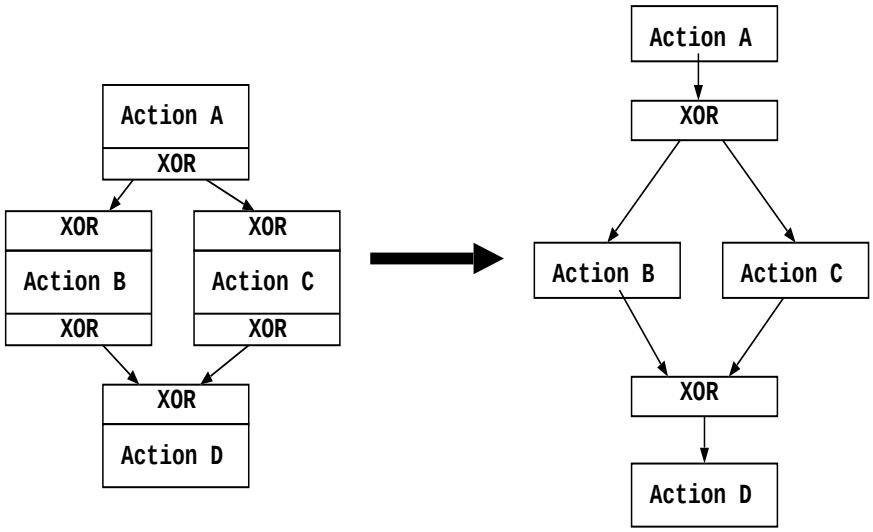

Fig. 1. Mapping a simple dependency graph to a graph with activity nodes and gateway nodes.

4 Discussion, conclusions and future work

In this paper, we have described how we are extending a framework for process mining and process comparison, whose first version has been published in [10]. In particular, we are making distance calculation more general, and closer to the semantic meaning of the mined process model, by explicitly taking into account gateway nodes. This enhancement can potentially represent a significant advance with respect to the state of the art research in the field, since gateway nodes are typically disregarded in process comparison approaches, with a few exception (see e.g., [7], which is however limited to identify changes with respect to an ongoing process instance, in an agile workflow context; see also [5], which makes gateway nodes explicit in the process model, but does not introduce a contribution related to them in the metric for process comparison).

The current approach will undergo a thorough analysis in the next months, and several experimental evaluations will be planned. We also wish to test our work in combination with different mining algorithms (other than heuristic miner). Finally, we plan to consider loops, a control flow structure which was never observed in the stroke management domain, but which could be significant in other applications.

We believe then, once extended and validated, the tool will represent a useful means for process comparison and retrieval, in medicine as well as in other application fields.

5 Ackowledgements

We are grateful to A. Cavallini and G. Micieli (IRCCS Fondazione "C. Mondino", Pavia, Italy) for providing stroke management data.

This research is partially supported by the GINSENG Project, Compagnia di San Paolo.

References

1. Workflow management coalition, *http : //www.wfmc.org/wfmc − publications.html*.
2. H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689694, 1997.
3. W. Van der Aalst, B. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters. Workflow mining: a survey of issues and approaches. *Data and Knowledge Engineering*, 47:237–267, 2003.
4. R. Dijkman, M. Dumas, and R. Garca-Banuelos. Graph matching algorithms for business process model similarity search. In *Proc. International Conference on Business Process Management*, pages 48–63, 2009.
5. R. Dijkman, B. Gfeller, J. Kuster, and H. Volzer. Identifying refactoring opportunities in process model repositories. *Information and Software Technology*, 53:937–948, 2011.

6. *http : //www.win.tue.nl/ieeetfpm*. IEEE Taskforce on Process Mining: Process Mining Manifesto.
7. M. Minor, A. Tartakovski, D. Schmalen, and R. Bergmann. Agile workflow technology and case-based change reuse for long-term processes. *International Journal of Intelligent Information Technologies*, 4(1):80–98, 2008.
8. S. Montani and G. Leonardi. Retrieval and clustering for supporting business process adjustment and analysis. *Information Systems*, DOI: http://dx.doi.org/10.1016/j.is.2012.11.006.
9. S. Montani and G. Leonardi. Retrieval and clustering for business process monitoring: results and improvements. In B. Diaz-Agudo and I. Watson, editors, *Proc. International Conference on Case-Based Reasoning (ICCBR) 2012, Lecture Notes in Artificial Intelligence 7466*, page 269283. Springer-Verlag, Berlin, 2012.
10. S. Montani, G. Leonardi, S. Quaglini, A. Cavallini, and G. Micieli. Mining and retrieving medical processes to assess the quality of care. In Sarah Jane Delany and Santiago Ontañón, editors, *ICCBR*, volume 7969 of *Lecture Notes in Computer Science*, pages 233–240. Springer, 2013.
11. G. Valiente. *Algorithms on Trees and Graphs*. Springer, 2002.
12. B. van Dongen, A. Alves De Medeiros, H. Verbeek, A. Weijters, and W. Van der Aalst. The proM framework: a new era in process mining tool support. In G. Ciardo and P. Darondeau, editors, *Knowledge Mangement and its Integrative Elements*, pages 444–454. Springer, 2005.
13. A. Weijters, W. Van der Aalst, and A. Alves de Medeiros. *Process Mining with the Heuristic Miner Algorithm, BETA Working Paper Series, WP 166*. Eindhoven University of Technology, Eindhoven, 2006.