Metric for Clustering Business Processes Based on Alpha Algorithm Relations

Fabio Aiolli, Andrea Burattin, and Alessandro Sperduti

Department of Pure and Applied Mathematics
University of Padua, Italy
{aiolli, burattin, sperduti}@math.unipd.it

Abstract. We present a metric for the comparison of business process models. This new metric is based on a representation of a given model as two sets of local relations between pairs of activities in the model. In order to build this two sets, the same relations defined for the Alpha algorithm are considered. The proposed metric is then applied to hierarchical clustering of business process models. A Process Log Generator tool has been implemented and made publicly available.

1 Introduction

Process mining algorithms designed for real world data typically cope with noisy or incomplete logs via techniques that require the analyst to set the value of several parameters. Because of that, many process models corresponding to different parameters settings can be generated, and the analyst very easily gets lost in such a variety of process models. In order for process mining algorithms to be really effective, it is of paramount importance to give to the analyst the possibility to easily interpret the output of the mining. In a technique for the automatic discretization of the space of the values of the parameters and a technique for selecting one among the potentially generable models have been proposed. Presenting just a single output model, however, could not be enough informative for the analyst, so the problem is how to find a way of presenting only a small set of informative results, so that the analyst can either point out the one that better fits the actual business context, or extract general knowledge about the business process from a set of relevant extracted models.

In order to pursue this objective, it is necessary to be able to compare different process models, so to avoid to present to the analyst too similar processes. In this work, we propose a model-to-model metric that allows the comparison between business processes, removing some of the problems which afflict other metrics already proposed in the literature. The proposed metric, in particular, transforms a given model into two sets of relations among process’ activities. The comparison of two models is then performed on the generated sets. In this

---

1 For details on PLG see and presents the PLG tool.

http://www.processmining.it/sw/plg Section 3.1
paper, we also shortly describe our implementation of the proposed metric, including a hierarchical clustering algorithm for business processes that can be easily derived from it.

The remainder of this paper is structured as follow: Section 2 presents the metric for the definition of the comparison metric between business process. A review of the literature is also presented, with the problem statement and the solution proposed. Section 3 proposes a possible way to exploit the metric presented: perform clustering on business processes and, in particular, how this approach can help users in finding “good results” for process mining. Section 5 concludes the paper and presents some possible future works.

2 Comparing processes

The comparison of two business processes is not trivial as it requires to select those perspective that should be considered relevant for the comparison. For example, we can have two processes having same structure (in terms of connections among activities) but different activity names. In this case, it is easy for a human analyst to detect the underlying similarity, while a machine will hardly be able to capture this feature unless previously programmed to do that. For this reason, several different comparison metrics have been developed in the recent past, each one focusing on a different aspect and related to a specific similarity measure.

2.1 A review of the literature

Comparison of processes has been the focus of several papers, especially in the context of process composition (e.g. in the case of web services), process diagnosis and conformance between a reference model and the result of a process mining control-flow discovery algorithm.

In the context of business process mining, one of the first works to propose a process metric are [4,15]. In those papers, process models are compared on the basis of typical behaviors (expressed as an event log). The underpinning idea is that models that differ on infrequent traces should be considered much more similar than models that differ on very frequent traces. Of course, this entails that a reference execution log is needed. In [12], the authors address the problem of detection of synonyms and homonyms that can occur when two business processes are compared. Specifically, a syntactic similarity is computed by comparing the number of characters of the activities names; linguistic similarity depends on a dictionary of terms and structural similarity is based on the hierarchical structure of an ontology. These three similarities are combined in a weighted average. The work by Bae et al. [5], explicitly refers to process mining as one of its purposes. The authors propose to represent a process via its corresponding dependency graph, which in turn is converted into its incidence matrix. The distance between two processes is then computed as the trace of $(N_1 - N_2) \times (N_1 - N_2)^T$, where $N_1$ and $N_2$ are the process incidence matrices.
The authors of [11] present an approach for the comparison of models on the basis of their “causal footprints”. A causal footprint can be seen as a collection of the essential behavioral constraints that a process model imposes. The similarity between processes is computed on the basis of their corresponding causal footprints, using the cosine similarity. Moreover, in order to avoid synonyms, a semantic similarity among function names is computed. The idea behind [10] is slightly different from the above-mentioned works as it tries to point out the differences between two processes so that a process analyst can understand them. Actually, this work is based on [11]. The proposed technique exploits the notion of complete trace equivalence in order to determine differences. The work by Wang et al. [18] considers only Petri nets. The basic idea is that the complete firing sequence of a Petri net might not be finite, so it is not possible to compare Petri nets in these terms. That’s why the Petri net is converted into the corresponding coverability tree (guaranteed to be finite) and the comparison is performed on the principal transition sequences, created from the corresponding coverability trees. The paper [20] describes a process in terms of its “Transition Adjacency Relations” (TAR). The set of TARs describing a process is a set of pairs of activities that occur one directly after the other. The TAR set of a process is always finite, so the similarity measure is computed between the TAR sets of the two processes. The similarity measure is defined as the ratio between the cardinality of the intersection of the TARs and the cardinality of the union of them. A recent work [19] proposes to measure the consistency between business processes representing them as “behavioral profiles” that are defined as the set of strict order, exclusiveness and interleaving relations. The approach for the generation of these sets is based on Petri nets (their firing sequences) and the consistency of two processes is calculated as the amount of shared holding relations, according to a correspondence relations, that maps transition of one process into transitions of the other.

2.2 Problem statement and the general approach

The first step of our approach is to convert a process model into another formalism where we can easily define a similarity measure. We think that the idea of [20], presented in the previous section, can be refined to better fit the case of business processes. In that work, a process is represented by a set of TARs. Specifically, given a Petri net $P$, and its set of transitions $T$, a TAR $\langle a, b \rangle$ (where $a, b \in T$) exists if and only if there is a trace $\sigma = t_1t_2t_3...t_n$ generated by $P$ and $\exists i \in \{1, 2, ..., n-1\}$ such that $t_i = a$ and $t_{i+1} = b$. For example, if we consider the two processes of Fig. 1, they have the same TAR sets: all the possible traces generated by them always start with the transition named “A” and end with “D”. In the middle, the process on the left hand side has two AND branches with the transitions “B” and “C” (so the TAR set must take into account all the possible combinations of their executions); the right hand side process has two XOR branches, and they describe all the possible combinations of the activities. Because of this peculiarity, the pairs of adjacent transitions that both...
process models can generate are the same, so their similarity measure is 1 (i.e. they describe the same process).

![Fig. 1: Two processes described as Petri nets that generate the same TAR sets. According to the work described in [20], their similarity would be 1, so they would be considered essentially as the same process.](image)

The main problem with this metric is that, even if from a “trace equivalence” point of view the two processes in Fig. 1 are the same (considering the two TAR sets), from a more practical (i.e. business processes) point of view they are not: e.g., the second process contains repeated activities and, more importantly, if activities “B” and “C” last for a certain time period (i.e. they are not instantaneous), then it is not the same to observe them in parallel or in (all the possible) sequences. Moreover, there are many processes that will generate the same set of traces and a metric for the comparison of processes should consider them as different. With the model-to-model metric presented in this work we try to address this problem.

Similarly to the cited work we also propose to first convert a process model from a (hard to work with) representation (such as a Petri net or an Heuristics net) into another (easier to handle) one; then the real comparison is performed on these new representations. However, in our case the model is transformed into two sets of relations instead of one. The comparison is then performed by combining the results obtained by the comparison of the two sets individually.

### 2.3 Process representation

As stated in the previous section our idea here is to convert a given process model into two sets: one set of relations between activities that must occur, and another set of relations that cannot occur. For example, consider the process in Fig. 2(a), where a representation of the process as a Petri net is given. That is a simple process that contains a parallel split in the middle. In Fig. 2(b), the same process is given but it is represented as a dependency graph.

In order to better understand the representation of business processes we are introducing, it is necessary to give the definition of workflow trace, i.e. the sequence of activities that are executed when a business process is followed. For example, considering again the process in Fig. 2 the set of all the possible traces that can be observed is

\[ \{ABCEFD, ABECFD, ABEFCD, AEBFCD, AEFCBD, AEFBCD\} \]
We propose to represent such kind of processes using two types of relations: a first set containing those relations that must hold, the second set containing those relations that cannot hold. Specifically, we consider relations ($A > B$ and $A \not> B$) which have been already used by the Alpha algorithm.

More formally, if a relation $A > B$ holds, it means that, in some workflow traces that the model can generate, activities $A$ and $B$ are adjacent: let $W$ be the set of all the possible traces of a model, then there exist at least one trace $\sigma = t_1 \ldots t_n \in W$, where $t_i = A$ and $t_{i+1} = B$ for some $i \in \{1, \ldots, n - 1\}$. The other relation $A \not> B$ is the negation of the previous one: if it holds, then, for any $\sigma = t_1 \ldots t_n \in W$, there is no $i$ for which $t_i = A$ and $t_{i+1} = B$. It is important to note that the above relations describe only local behaviors (i.e., they do not consider activities that occur far apart). Moreover, it must be noticed that our definition of $>$ is the same as the one used in [20].

These relations have been presented in [2,14,3] and are used by the Alpha algorithm for calculating the possible causal dependency between two activities. However, in that case the idea is different: given a workflow log $W$, the Alpha algorithm finds all the $>$ relations and then, according to some predefined rules, these relations are combined to get more useful derived relations. The particular rules which are mined starting from $>$ are:

1. $A \rightarrow B$, iif $A > B$ and $B \not> A$;
2. $A \# B$, iif $A \not> B$ and $B \not> A$;
3. $A \parallel B$, iif $A > B$ and $B > A$.

Here, the relations $>$ and $\not>$ will be called primitive relations, while $\rightarrow$, $\#$ and $\parallel$ will be called derived relations. The basic ideas underpinning these three rules are that (1) if two activities are observed always adjacent and in the same order, then there should be causal dependency between them ($\rightarrow$); (2) if two activities are never seen as adjacent activities, it is possible that they are not in any causal dependency ($\#$) (3) if two activities are observed in no specific order, it is possible that they are in parallel branches ($\parallel$). Starting from the given definition, it is clear that, given two activities contained in a log, at most one derived relation ($\rightarrow$, $\#$ and $\parallel$) can hold between them. In particular, if these two activities appear adjacent in the log, then one of these relations holds, otherwise, if they are far apart, none of the relations hold.
The idea is to perform a “reverse engineering” of a process in order to discover which relations must be observed in an ideal “complete log” (a log containing all the possible behaviors) and which relations cannot be observed. The Alpha algorithm describes how to mine a workflow log to extract sets of holding relations that are then combined and converted into a Petri net. The reverse approach can be applied too, even if it is less intuitive. So, the idea presented in this work is to convert a Petri net into a set of \( > \) and \( \not> \) relations.

To further understand our approach, it is useful to point out the main differences with respect to the Alpha algorithm. Considering Fig. 3, filled lines represent what the Alpha algorithm does: starting from the log (i.e. the set of traces) it extracts the primitive relations that are then converted into derived relations and finally into a Petri net model. In our approach that procedure is reversed and is represented with dotted lines: starting from a given model (Petri net or dependency graph, or any other process model), derived relations are first extracted and then converted into primitive ones; the comparison between business process models is actually performed at this level.

![Diagram](https://via.placeholder.com/150)

**Fig. 3:** Representation of the space where the comparison between processes is performed. The filled lines represent the steps that are performed by the Alpha algorithm. The dotted lines represents the conversion of the process into sets of primitive relations, as presented in this work.

Note that, since the naive comparison of trace equivalence is not feasible (in case of loops, the generation of the trace could never stop), we decided to analyze a model (e.g. a Petri net or an Heuristics net) and see which relations can possibly be derived. Given the set of derived relations for a model, these will be converted into two sets of positive and negative relations.

The main difference with other approaches in the literature ([19][20]), is that our approach can be applied on every modeling language and not only Petri net or Workflow net. This is why our approach cannot rely on Petri net specific notions (such as firing sequence). We prefer to just analyze the structure of the process from a “topological” point of view. In order to face this problem, we decided to consider a process in terms of composition of well known patterns.
Right now, a small but very expressive set of “workflow patterns” [17] are taken into account. These patterns are the ones presented in Fig. 4.

When a model is analyzed, the following derived relations are extracted:

- a sequence of two activities $A$ and $B$ (Fig. 4a), will generate a relation $A \rightarrow B$;
- every time an XOR split is observed (Fig. 4d) and activities $A$, $B$ and $C$ are involved, the following rules can be extracted: $A \rightarrow B$, $A \rightarrow C$ and $B \neq C$; a similar approach can handle the XOR join (Fig. 4e), generating a similar set of relations: $D \rightarrow F$, $E \rightarrow F$, $D \neq E$;
- every time an AND split is observed and activities $A$, $B$ and $C$ are involved (Fig. 4b), the following rules can be extracted: $A \rightarrow B$, $A \rightarrow C$ and $B \parallel C$; a similar approach can handle the AND join (Fig. 4c), generating a similar set of relations: $D \rightarrow F$, $E \rightarrow F$, $D \parallel E$.

For the case of dependency graphs, this approach is formalized in Algorithm 1, the basic idea being that given two activities $A$ and $B$, directly connected with an edge, the relation $A \rightarrow B$ must hold. If $A$ has more than one outgoing or incoming edges ($C_1, \ldots, C_n$) then the following relations will also hold: $C_1 \rho C_2, \ldots, C_1 \rho C_n, \ldots, C_n - 1 \rho C_n$ (where $\rho$ is $\neq$ if $A$ is a XOR split/join, $\rho$ is $\parallel$ if $A$ is an AND split/join).

Once the algorithm has completed the generation of the set of holding relations, this can be split in two sets of positive and negative relations, according to the “derived relations” presented in Section 2.3. Just to recap, we have that
Algorithm 1: Conversion of a dependency graph into sets of relations.

```
input : G = (V, E): process as a dependency graph
T : V → {XOR split, XOR join, AND split, AND join}
output: R: set of holding relations

1. foreach (v₁, v₂) ∈ E do
   2.     R = R ∪ {v₁ → v₂};
3. end

4. foreach v ∈ V, X = {u ∈ V | (v, u) ∈ E} do
5.     foreach (u₁, u₂) ∈ X × X such that u₁ ≠ u₂ do
6.         if T(v) is XOR split then
7.             R = R ∪ {u₁#u₂}
8.         else if T(v) is AND split then
9.             R = R ∪ {u₁∥u₂}
10.        end
11.    end
12. end

13. foreach v ∈ V, X = {u ∈ V | (u, v) ∈ E} do
14.     foreach (u₁, u₂) ∈ X × X such that u₁ ≠ u₂ do
15.         if T(v) is XOR join then
16.             R = R ∪ {u₁#u₂}
17.         else if T(v) is AND join then
18.             R = R ∪ {u₁∥u₂}
19.         end
20.    end
21. end

22. return convertRelations(R) /* to convert the set of derived relations R into two sets of primitive relations */
```

A → B generates A > B and B ≠ A; A#B generates A ≠ B and B ≠ B; and, finally, A∥B generates A > B and B > A.

Let’s consider again the process P of Fig. 2. After the execution of the three “foreaches” in Algorithm 1 (so before the return of the last line), R will contain all the derived relations that, in the considered example, are:

```
A → B  A → E  B → C  E → F  C → D  F → D  B∥E  C∥F
```

These will be converted during the return operation of the algorithm into these two sets:

```
R⁺(P) = {A > B, A > E, B > C, E > F, C > D, F > D, B > E, E > B, C > F, F > C}
```

```
R⁻(P) = {B ≠ A, E ≠ A, C ≠ B, F ≠ E, D ≠ C, D ≠ F}
```

It is important to maintain these two sets separated because of the metric, that will be presented in Section 2.4.
2.4 A Metric for processes comparison

Converting a process model into another representation is useful to compare two processes in a more easy and effective way. Here we propose a way to use the previously defined representations to obtain a principled metric. Specifically, given two processes $P_1$ and $P_2$, expressed in terms of positive and negative constraints: $P_1 = (R^+, R^-)$ and $P_2 = (R^+, R^-)$ they are compared according to the amount of shared “required” and “prohibited” behaviors. A possible way to compare these values is the Jaccard similarity $J$ and the corresponding distance $J_δ$, that is defined, between two sets, as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad J_δ(A, B) = 1 - J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

in [16], for example, it is proven that the Jaccard is actually a distance measure over sets (so it is non-negative, symmetric and satisfies the identity of indiscernibles and the triangle inequality).

Our new metric is built considering the convex combination of the Jaccard distance for the set of positive and negative relations of two processes:

$$d(P_1, P_2) = \alpha J_δ (R^+(P_1), R^+(P_2)) + (1 - \alpha) J_δ (R^-(P_1), R^-(P_2))$$

where $0 \leq \alpha \leq 1$ is a weighting factor that allows the user to calibrate the importance of the positive and negative relations. Since this metric is defined as a linear combination of distances ($J_δ$), it is a distance itself. It is important that the given measure is actually a metric, because the final aim of this work is doing clustering on those business processes.

It is important to note that there are couples of relations that are not “allowed” at the same time, otherwise the process is ill-defined and shows problematic behaviors, e.g. deadlock\textsuperscript{2}. Incompatible couples are defined as follows:

- if $A \rightarrow B$ holds then $A \parallel B$, $B \parallel A$, $A\#B$, $B\#A$, $B \rightarrow A$ are not allowed;
- if $A \parallel B$ holds then $A\#B$, $B\#A$, $A \rightarrow B$, $B \rightarrow A$, $B \parallel A$ are not allowed;
- if $A \#B$ holds then $A \parallel B$, $B \parallel A$, $A \rightarrow B$, $B \rightarrow A$, $B \#A$ are not allowed.

Similarly, considering primitive relations, if $A > B$ holds then $A \not> B$ represents an inconsistency so this behavior should not be allowed.

**Theorem 1.** Two processes composed of different patterns, that do not contain duplicated activities and that do not have contradictions into their set of relations (either derived or primitive), have distance measure greater than 0.

*Proof.* Since the distance measure is calculated on the basis of the two sets of primitive relations, two processes $P_1 = (R^+_{P_1}, R^-_{P_1})$ and $P_2 = (R^+_{P_2}, R^-_{P_2})$ have a distance measure $d(P_1, P_2) > 0$ if the sets $R^+_{P_1}$, $R^-_{P_1}$ and $R^+_{P_2}$, $R^-_{P_2}$ are not pairwise equal. The two sets $R^+$ and $R^-$ are generated starting from the derived relations are present at the same time.

\textsuperscript{2} It must be stressed that a process may be ill-defined even if no such couples of relations are present at the same time.
relations, and these are created starting from the patterns observed. If we assume that two processes are made of different patterns, they will generate different sets of derived relations and thus different sets of primitive relations. This is going to generate a distance measure, for the two processes that is greater than 0.

Since the sets of relations are generated without looking at the set of traces, but just starting from the local structure of the process model, if it is not sound (considering the Petri net notion of soundness) it is possible to have “contradictions”.

There is another important aspect that needs to be pointed out: in the case of contradictions, there may be an unexpected behavior of the proposed metric. However, in the case of contradictions, there can be unexpected behavior. For example, in Fig. 5, the two processes are “structurally different”, but have distance measure equals to 0. This is due to the contradictions contained in the set of primitive relations that are generated because of the contradictions on the derived relations (in both processes $B || C$ and $B \# C$ hold at the same time). More generally, we have that two different processes have distance measure equals to 0 when their differences results in contradictions.

Consider the three processes of Fig. 1a, 1b and 2. Table 1 proposes the values of the TAR metric [20], compared with the ones of the metric proposed in this work, with different values of its parameter $\alpha$. Note that, when $\alpha = 1$ then only the positive relations are considered; when $\alpha = 0$, only negative relations are taken into account; and, when $\alpha = 0.5$, the two cases are equally balanced. Moreover, in the situation presented here, the TAR metric and the metric of this work (with $\alpha = 1$) are equal but, generally, this is not the case (when there is some concurrent behavior, TAR metric adds relations with all the other activities in the other branches, whereas our metric adds only local relations with the firsts activities of the branches).

3 Clustering processes using model-to-model metric

Once the metric on business processes is available, it is possible to perform many operations that exploits such function. The one we decided to do as first is clustering business processes. Since the space we are dealing with is unknown, we decided to use an agglomerative hierarchical clustering algorithm [13] with,
in this first stage, an average linkage (or average inter-similarity): in this case the similarity $s$ between two clusters, $c_1$ and $c_2$, is defined as the similarity of all the pairs of activities belonging to the two clusters:

$$s(c_1, c_2) = \frac{1}{|c_1| |c_2|} \sum_{p_i \in c_1} \sum_{p_j \in c_2} d(p_i, p_j)$$

The basic idea of agglomerative hierarchical clustering is to start with each element in a singleton cluster and, at each iteration of the algorithm, the two closest clusters are merged into one. The procedure iterates until a single cluster is created, containing all the elements. The typical way of representing a hierarchical clustering is using a dendrogram, that represents how the elements are combined together. Fig. 6 represents a set of dendrogram obtained with our clustering of business processes.

### 3.1 Implementation in the PLG software

The entire procedure has been implemented in PLG\(^3\) [8], a software for the generation of random business processes. Fig. 6 reports dendrograms with $\alpha \in \{0, 0.5, 1\}$ as presented by the widget that has been implemented. It is divided in two parts: the left hand side proposes a graphical representation of the similarity matrix, where the similarity is expressed in terms of colors (a black cell means that the similarity is 0, a red cell means similarity 1). The right hand side of the widget presents the actual dendrogram (with the similarity between each pair of clusters). Hierarchical clustering has been performed on ten processes which have been randomly generated by the software. The result is presented in the figure. In the lower part of the same figure examples of two processes considered “distant” are also reported.

### 3.2 Exploitation of clustering for process mining

A possible way to exploit the clustering technique proposed in this work is to allow absolute-non-expert analyst to perform process mining. In particular, as

---

\(^3\) The PLG software is a free and open source software and can be downloaded at [http://www.processmining.it/sw/plg](http://www.processmining.it/sw/plg)
(a) Dendrograms generated with $\alpha = 0$.  
(b) Dendrograms generated with $\alpha = 1$.  
(c) Dendrograms generated with $\alpha = 0.5$.  
(d) Process 4, as presented in the dendrograms above.  
(e) Process 10, as presented in the dendrograms above.  

Fig. 6: The topmost figures represent three dendrograms. The two Petri nets, are two examples of “distant” processes.
stated in [6], non-expert users can have many problems in the configuration of the parameters for the mining (these parameters can be real-valued and there can be no evidence on their contribution on the results). Our approach shifts the problem from choosing the best parameters configuration to selecting the model that better describe the actual process performed. This is the main reason why such approach can also be called “parameter configuration by result exploration”. The idea can be split in the following steps:

1. the system receives a log as input;
2. the space of the parameters can be discretized in order to consider only the meaningful values (from an infinite space to a finite one);
3. all the distinct models that can be generated starting from the parameters are generated, so to have an exhaustive list of all the models that can be inferred starting from the log;
4. all the generated processes are clustered (using the technique, previously presented, in Section 3);
5. the hierarchy of clusters is “explored” by the user by drilling down on the direction that he/she thinks being the most promising one.

A practical example of the given approach is presented in the following section.

4 Clustering for process mining

As presented in the previous sections, our idea is to use the clustering of business processes to allow non expert users to perform process mining (as control-flow discovery). A proof of concept procedure has been implemented in PLG and is available for the download.

The approach has been tested on a process log with 100 cases and 46 event classes, equally distributed among each case, with 2 event types. The complete set of possible business processes is made of 783 models that are generated starting from the possible configurations of the algorithm Heuristics Miner++[7].

A complete representation of the clusters generated from such dataset has not been created because of problems in exporting the image, however, a representation of a subset of them (350 process models) is proposed in Fig. 7. This is a dendrogram representation of the hierarchy that comes out of the distance function presented in previous sections. The distance matrix, with distances per each pair of models, is presented in Fig. 8.

As previously stated, a proof of concept has been implemented in PLG and a screenshot of the component is presented in Fig. 9. The idea is to start from the “root” of the dendrogram and navigate it until a leaf is reached. Since a dendrogram is a binary tree, every cluster is made of two sub-clusters that are represented by their corresponding medoids. These two process models (i.e. the medoids) are proposed to the user that can decide which is the best “direction” to follow, in order to extract the model that is the most realistic one.

There can be dependencies among parameters, so that changing the value of one of them does not necessarily turn out in a different result.
Fig. 7: Dendrogram generated starting from the distance matrix of Fig. 8.

The idea is that, in the initial steps, the user will be asked to select between models that are very different each other. So, the as long as the user makes decisions, the processes to compare will be closer each other, so the user decision can be based on other aspects.

5 Conclusions and future work

This work presented a new approach for the comparison of business processes. This approach relies on the conversion of a process model into two sets of relations: the first contains all the local relations (only between two connected activities) that must hold; the second with the relations that must not hold. These two sets are generated starting from the relations of the Alpha algorithm but, instead of starting from a log and performing abstractions to achieve some rules, the opposite way is followed: given the model, local relations (expressed in terms of behavior that is allowed in the log trace) are extracted. The proposed metric is based on the comparison of these two sets.

In future work, to extend what has already been done, are already planned: first of all, some deeper insights on the case of contradictory relations (e.g. when both $A > B$ and $A \not> B$ hold) seems to be required. Moreover, there are some new ideas that might be useful, for example: considering not only sets of primitive relations, but multisets (so the case where cardinality is not in $\{0,1\}$ but it is $\mathbb{N}$) of relations and add, to the distance measure, also the distance between the labels of the activities (synonyms will be counted as the same activity).
Fig. 8: Distance matrix of 350 process models, generated as different configuration of the Heuristics Miner++ parameters. The brighter an area is (e.g. the diagonal), the higher is the similarity between the two processes.
Fig. 9: The window for the exploration of the dendrogram. In this case, the user has to choose between two clusters (whose medoids are “Process 7” and “Process 14”). Their corresponding children are presented too, just to help the analyst in finding the best direction.

From an implementation point of view, it seems very promising to let the user explore the space of the generated processes of [6] through the medoids of the clusters resulting as output of the generation of all the models, given one log, and changing the parameter values for the mining. In particular, it seems that the distance measure can be computed very quickly in the case of the Heuristics Miner algorithm, given the same log but different mining parameters.

Acknowledgements This work was supported by SIAV S.p.A.

References