A Co-Training Strategy for Multiple View Clustering in Process Mining

Annalisa Appice and Donato Malerba, *Member, IEEE*

Abstract—Process mining refers to the discovery, conformance, and enhancement of process models from event logs currently produced by several information systems (e.g. workflow management systems). By tightly coupling event logs and process models, process mining makes it possible to detect deviations, predict delays, support decision making, and recommend process redesigns. Event logs are data sets containing the executions (called traces) of a business process. Several process mining algorithms have been defined to mine event logs and deliver valuable models (e.g. Petri nets) of how logged processes are being executed. However, they often generate spaghetti-like process models, which can be hard to understand. This is caused by the inherent complexity of real-life processes, which tend to be less structured and more flexible than what the stakeholders typically expect. In particular, spaghetti-like process models are discovered when all possible behaviors are shown in a single model as a result of considering the set of traces in the event log all at once.To minimize this problem, trace clustering can be used as a preprocessing step. It splits up an event log into clusters of similar traces, so as to handle variability in the recorded behavior and facilitate process model discovery. In this paper, we investigate a multiple view aware approach to trace clustering, based on a co-training strategy. In an assessment, using benchmark event logs, we show that the presented algorithm is able to discover a clustering pattern of the log, such that related traces result appropriately clustered. We evaluate the significance of the formed clusters using established machine learning and process mining metrics.

Index Terms—Clustering, co-training, multiple view learning, process mining

1 Introduction

Information systems are becoming more and more intertwined with the operational processes they support. As a result, a huge volume of events is recorded by several of today's enterprise systems. Nevertheless, organizations still have problems extracting useful information on process executions, by analyzing this abundance of event data. The goal of process mining is to extract process-related information by observing events recorded in event logs [1]. An event log is a bag of process executions of a business process. A process execution is recorded as a trace. A trace is defined as an ordered list of activities invoked by a process execution from the beginning of its execution to the end. Process mining can be used to measure the conformance of traces to a prescribed process model or to enhance a known process model with additional information extracted from event logs. However, the crucial role of process mining is discovering abstract process models (e.g. Petri nets) from event logs. These models, which may also be extracted incrementally [2], can then be used for a variety of reasons, when redesigning processes and introducing new information systems. In particular, they can be used for discussion, documentation, performance analysis, as well as the enactment of the actual process executions.

Discovering the actual process that is being executed may be particularly difficult in several real-life environments, due to the highly flexible, complex nature of processes, which tend to be less structured than the stakeholders typically expect. Healthcare, product development and customer support are some of the examples of these flexible environments. Process mining algorithms may have problems dealing with such unstructured processes and generate spaghetti-like process models [3], which are hard to understand. This is caused by the inherent complexity of processes, while all possible behaviors are shown in a single diagram. An approach to overcome this limit is to cluster the traces, such that each of the resulting clusters corresponds to coherent sets of related process executions, which can each be significantly represented by a process model.

Traditional studies on trace clustering can be classified in two categories: (1) Studies that operate on the traces transformed into a vector space model [3], [4], [5]. Here clustering is done using distance-based machine learning algorithms with different metrics (e.g. euclidean distance and Jaccard distance) in the vector space. (2) Studies that operate on the traces as-is, by applying clustering algorithms with sequence distance metrics (e.g. Levensthein distance) [6], [7], [8]. In this paper, we account for considerations reported in [9] and resort to a vector space approach to address the trace clustering problem. Log traces are mapped into a feature space, where computationally efficient vector-based algorithms can be used.

The main issue of a vector space approach is the definition of the vector space model. As discussed by van der Aalst [1], event logs contain a wealth of information related to several perspectives, such as the control-flow perspective (ordering of activities), the organizational perspective (organization of resources), the trace perspective (frequency of activities) and the performance perspective (time performances). While

• *The authors are with the Dipartimento di Informatica, Università degli Studi Aldo Moro di Bari, via Orabona, 4 - 70125, Bari, Italy. E-mail: {annalisa.appice, donato.malerba}@uniba.it.*

Manuscript received 9 Jan. 2015; revised 17 Apr. 2015; accepted 28 Apr. 2015. Date of publication 6 May 2015; date of current version 9 Dec. 2016.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2015.2430327

the main focus of the control-flow data is on modeling constructs that can be identified during process modeling, the organizational data can be used to gain insight into typical work patterns and organizational structures, the activity data can be used to better understand decision making and analyze differences between traces, timestamps can be used to diagnoze performance-related problems (e.g. bottlenecks). It is noteworthy that these different perspectives may generate distinct vector space models of traces. These are called trace profiles (or views). Song et al. [5] have defined features associated to these trace profiles and considered these features to address the trace clustering problem. However, they have used each trace profile independently, in order to compute a profile-specific clustering pattern of the traces processed. More recently, van der Aalst [1] has promoted the idea that these different perspectives can also be merged into a single model, providing an integrated view of the process, in order to perform "what if" analysis using simulation. We decided to follow this idea and address the trace clustering problem by exploring and exploiting these multiple profiles simultaneously. However, it is improper to concatenate features of several profiles into one long vector, because they have different properties, and simply concatenating them into a high-dimensional feature vector will suffer from the so-called curse of dimensionality [10]. The multiple view clustering approach is investigated here, in order to improve process discovery by a better cluster partition of the traces than single view clustering.

Multiple view clustering [11], [12], [13] has become popular in recent years. It is considered a viable solution to the curse of dimensionality problem in classification when multiple views of data are available [10]. However, to the best of our knowledge, it has never been applied to process tasks. Thus, we have decided to investigate multiple view clustering in process mining, in order to bridge the gap between the clustering bias and the choice of the process mining perspective bias from which traditional clustering approaches may suffer. We discover a consensus clustering pattern of the processed traces, so to deal with possible curse of dimensionality issue, when processing multiple perspectives (profiles) of data simultaneously. We work on the assumption that the truth underlying the clustering pattern would assign a trace to the same cluster independently of the trace profile. Therefore, we decide to use simultaneously all trace profiles by bootstrapping the clusterings of different profiles using information from one another. This is done with a co-training strategy [14], in order to constrain the search for the clusterings that agree across the profiles.

Co-training was originally introduced (and is still mostly used) in the setting of semi-supervised learning [15], [16], [17] to process datasets, whose features are separated into two (or more) disjoint sets, which are regarded as independent data profiles. In this paper, we take the co-training approach to an unsupervised learning setting. Unsupervised co-training was already investigated in [18], for general-purpose multiple view spectral clustering, but here it is adapted to the event log context. At each iteration of the unsupervised co-training process, multiple trace clusterings are trained independently from multiple trace profiles. The clustering from one profile is used to constrain the similarities used for the other profiles. By iteratively applying this approach, clusterings of considered profiles converge towards a unique clustering pattern.

TABLE 1
A Fragment of an Example Event Log

TraceId	Activity	Timestamp	Resource
1	Register request (R)	2010-12-30:11:02	Pete
1	Examine throughly (ET)	2010-12-31:10:06	Sue
1	Check ticket (CT)	2011-01-05:15:12	Mike
1	Decide (D)	2011-01-06:11:18	Sara
1	Reject request (RR)	2011-01-07:14:24	Pete
2	Register request (R)	2010-12-30:11:32	Mike
2	Check ticket (CT)	2010-12-30:12:12	Mike
2	Examine causally (EC)	2010-12-30:14:16	Pete
2	Decide (D)	2011-01-05:11:22	Sara
2	Pay compensation (PC)	2011-01-08:12:05	Ellen
3	...	...	...

Each event is linked to a specific trace. It corresponds to an activity, has a timestamp and is triggered by a resource.

The specific contributions of this paper are in: (1) The use of several trace profiles in multiple view learning. (2) The development of a multiple trace profile aware clustering algorithm, which adapts the iterative co-training strategy, originally defined in [18], to the process mining setting. We formulate a general co-training strategy that can be run with any distance-based partitioning algorithm. We use multiple (more than two) trace profiles, in order to compute clustered traces. We use the Silhouette width to formulate a stopping criterion of the iterative procedure. (3) An extensive evaluation of the effectiveness of the proposed approach on trace clustering problems in several benchmark event logs. The paper is organized as follows. Section 2 reports preliminary concepts. Section 3 describes related work. Section 4 illustrates the proposed algorithm, while Section 5 reports the analysis of its time complexity. Section 6 describes the datasets, the experimental setup and discusses the relevant results. Finally, Section 7 draws some conclusions and outlines some future work.

2 Preliminary Concepts

In this section, we report the basic concepts of event, trace, event log, profile and feature, as well as introduce the ideas behind multiple view clustering and trace clustering.

The basic assumption is that the event log contains information on activities executed for specific traces of a certain process type, as well as their resources and durations. An *event* ϵ is characterized by a set of mandatory characteristics, that is, the event corresponds to an activity, has a timestamp which represents date and time of occurrence and is triggered by a resource. An event log is a set of events. Each event in the log is linked to a particular trace and is globally unique. A *trace* T represents the execution of a business process instance. It is a finite sequence of distinct events, such that time is non-decreasing in the trace (i.e. for $1 \leq i < j \leq length(T) : timestamp(\epsilon_i) \leq timestamp(\epsilon_j)$). An *event log* $\mathcal{L}$ is a bag of traces. An example reporting a fragment of an event log is reported in Table 1.

The traces of an event log can be characterized by profiles, where a *profile* is a vector of features (vector space model) that describe the traces from a specific perspective.

Every *feature* corresponds to a metric that assigns a numeric value to each trace. Thus, profiling an event log corresponds to defining a function that assigns each trace of the event log with a vector of values, measured one for each profile feature. The resulting vectors can be used to calculate the similarity between any couple of traces, using a vector space distance metric.

Multiple view learning is a direction in machine learning with good theoretical underpinnings and great practical success [19]. The main challenge of multiple view learning is to develop algorithms that use multiple data views (i.e. profiles) simultaneously, given the diversity of the feature set in each view. In *clustering*, it exploits multiple views, which could also have very different statistical properties, in order to find groups of examples that conform to similar behavior with respect to the *several* data views considered.

Trace clustering is used to separate traces into different groups, for which a more accurate and comprehensible process model can be discovered. Trace clustering is not different from regular clustering, aside from the input: an event log which is a collection of traces.

3 Related Works

Several studies in recent literature concern either multiple view clustering or trace clustering.

3.1 Multiple View Clustering

Long et al. [20] identify two directions for seeking solutions to multiple view clustering: centralized and distributed. Centralized algorithms make use of multiple representations simultaneously, in order to mine directly a single partitioning of the data. Distributed algorithms make use of multiple representations separately, in order to learn individual clustering patterns from each separate representation and, subsequently, combine the individual clusterings to produce a final single partitioning.

In the centralized framework, studies in multiple view clustering mainly extend well-known clustering algorithms, in order to use multiple independent feature sets simultaneously. Bickel and Scheffer [11] develop both a two-view EM and a two-view k-means algorithm under the assumption that the views are independent. de Sa [21] defines a two-view spectral clustering algorithm that, based on the minimizing-disagreement idea, creates a bipartite graph of the views. This algorithm also assumes that the two views are independent. Long et al. [22] propose a general model for multiple view clustering, which handles more than two views and representations from both vector and graph spaces. Zhou and Burges [23] define an algorithm that generalizes the single view normalized cut to the multiple view case. Tzortzis and Likas [24] present a multiple view clustering algorithm based on the convex mixture model. Cleuziou et al. [25] propose a collaborative approach, based on fuzzy k-means, which aims at minimizing the inertia of the fuzzy clusters in each view and penalizing the disagreement between any pairs of views. Kumar and Daumé [18] formulate a multiple view clustering algorithm in the co-training learning style. They use co-training, in order to fit two views through a spectral clustering algorithm. Kumar and Daumé [18] use the spectral information from one view to constrain the similarity graph used for the other view. They show that, by iteratively applying this approach an a-priori defined number of times, the clusterings of the two views tend towards each other. Tao et al. [26] investigate a variant of the co-training algorithm described in [18], which integrates spectral embedded clustering.

In the distributed framework, studies in multiple view clustering mainly investigate the idea of defining a function, in order to make the clusterings computed from different model spaces comparable. Long et al. [20] define several cluster mapping functions that look for the optimal clustering pattern from multiple models of multiple representations. It is noteworthy that this idea of looking for mapping multiple clusterings into a singe model is actually investigated in clustering ensemble learning. In particular, ensembles combine different component clusterings, in order to compute a better final partition, via a consensus function. Strehl and Ghosh [27] define three different consensus functions: the cluster-based similarity function, the hyper-graph partitioning algorithm and meta-clustering. Cheng and Zhao [28] apply clustering ensembles to multiple view learning problems. Fred and Jain [29] define a split-and-merge ensemble strategy for multiple view clustering. They compute single-view clustering patterns (split) and use them to generate view-aware cluster features (i.e. whether an example belongs to a cluster or not) of a new consensus clustering problem (merge). In particular, the single-view clusterings are combined in a consensus pattern by clustering data described by the cluster-based features originating from the multiple views processed. Chaudhuri et al. [12] describe a subspace learning algorithm based on Canonical Correlation Analysis, in order to generate a cluster pattern from uncorrelated data views. Cai et al. [13] derive a large-scale multiple view k-means clustering algorithm, which can be parallelized and performed on multi-core processors for large data analysis.

3.2 Trace Clustering

Several approaches to compute clustered traces are investigated in the process mining literature. Many studies apply a kind of translation of the event log into a vector space model, such that existing vector space distance metrics can be computed between each couple of traces and existing distance-based clustering algorithms can be performed. Alternatively, algorithms exist that define the distance between two traces without translation into a vector space context. Finally, there are model-based clustering algorithms that are formulated for trace clustering problems as well.

Trace clustering problems are investigated in [4], where Greco et al. formulate an iterative hierarchical algorithm to refine the process model. They use a vector space model considering both the activities and their transitions, in order to cluster the traces of an event log. They adopt the k-means algorithm as a base clustering algorithm. Song et al. [5] formulate a multitude of so-called trace profiles, each one can be selected, in order to determine a vector space model associated with the traces of an event log. Although they define many trace profiles, they use each profile independently for computing profile-specific clustered traces. Bose and Aalst [3] extend sequence-based learning approaches, in order to improve the way the control-flow context information is taken into account to construct the vector space model. They find sub-sequences of activities, also of different lengths, which are conserved across traces and use these conserved models for constructing a vector space model for the traces of the event log. Then they use agglomerative clustering as the underlying clustering algorithm. It is noteworthy that context-aware refers to control-flow properties of the traces of the event log, while it neglects contextual information referring to recourses, trace performances, and so on.

Model-based approaches are investigated in [6], where Bose and van der Aalst propose a trace-defined edit distance that is founded on the Levenshtein distance. They rely on deriving specific substitution, insertion and deletion costs, so as to take into account the transition behavior of the traces without any translation of the event log. They evaluate the edit distance with agglomerative clustering. Ferreira et al. [7] formulate a trace clustering algorithm that learns a mixture of first-order Markov chains using the expectation-maximization algorithm. The probability that a trace belongs to a cluster is estimated as the probability that the observed trace is produced by the Markov chain associated with that cluster. De Weerdt et al. [8] show how this approach may suffer from scalability problems. They present a trace clustering algorithm that is explicitly defined for finding an optimal distribution of traces, so as to maximize the combined accuracy of the underlying models. They employ an active learning inspired approach that centers on optimizing the process model accuracy. However, in their approach, clusters strongly depend on the conformance measures used for evaluating the accuracy of process models, as well as the algorithms considered for the process model discovery.

4 Co-Training Based Clustering

In this section, we describe a vector space trace clustering algorithm, called CoTraDiC (CO-training based TRAce DIstance-based Clustering), which is formulated in centralized multiple view learning by resorting to the co-training strategy. It allows us to draw a consensus trace clustering pattern that integrates the trace profiles of the multiple process mining perspectives of an event log. The considered profiles are those related to the data perspectives mainly investigated in process mining. These are the control-flow perspective, the organizational perspective, the trace perspective and the performance perspective [1]. As these perspectives convey information related to different entities (control-flow, resource, activity and timestamps), we can reasonably assume that they produce trace profiles that are compatible with the assumption of view independence, that is generally done in co-training [14]. We formulate the co-training strategy for distance-based clustering algorithms. This category of algorithms, which determine clusters by optimizing a distance-based criterion function, is frequently employed in process mining [3], [4], [5], [6], [7], [8], as the distance is, in general, easily defined and computed. Additionally, distance-based clustering algorithms usually need just one parameter, that is, either the number of final clusters or the minimum distance to separate two clusters [30]. The clustering algorithm is run simultaneously on the multitude of trace profiles of the event log, while the co-training is used to bootstrap the computation of the (dis-)similarity matrix of a profile, using clustering information from every other profile. It is noteworthy that the defined co-training strategy is independent of the number of profiles processed, the distance measure computed, as well as the distance-based clustering algorithm performed. Trace profiles, the co-training strategy, as well as clustering patterns discovered by co-training are described in the following sections.

4.1 Trace Profiles

Based on the data perspectives investigated in process mining, Song et al. [5] define a set of features to describe the event logs in each perspective. These features populate the trace profiles used in this work.

The *Activity profile* is defined from the trace perspective, that focuses on the activities of a trace [1]. It constructs one feature per type of activity found in the event log. Each activity feature is measured by counting the number of events of the trace having the specified activity's name.

Example 4.1 (Activity profile). Let us consider the event log in Table 1. It collects events for seven distinct types of activity, namely Register request (R), Examine thoroughly (ET), Check ticket (CT), Decide (D), Reject Request (RR), Examine casually (EC) and Pay compensation (PC). Each trace will be translated into a vector of seven feature values, one for each distinct type of activity. These features measure the number of times a specific type of activity is repeated in the trace. They are defined as follows:

TraceId	R	ET	CT	D	RR	EC	PC
1	1	1	1	1	1	0	0
2	1	0	1	1	0	1	1

The *Resource profile* is defined from the organizational perspective, that focuses on the resources of a trace [1]. It defines one feature per resource found in the event log. Each resource feature is measured by counting the number of events of the trace triggered by the specified resource.

Example 4.2 (Resource profile). Let us consider the event log in Table 1. It collects events for five distinct resources, namely Pete, Sue, Mike, Sara and Ellen. Each trace will be translated into a vector of five feature values, one for each distinct resource. These features measure the number of times a specific resource participate in an event of the trace. They are defined as follows:

TraceId	Pete	Sue	Mike	Sara	Ellen
1	2	1	1	1	0
2	1	0	2	1	1

The *Performance profile* is defined from the performance perspective, that focuses on the timestamps of a trace [1]. It has a predefined set of features, that is, the size of the trace (i.e. the number of events in the trace), the time duration of the trace (i.e. the time difference computed between the last event in the trace and the first event in the trace), as well as the minimum, maximum, mean and median time difference computed between consecutive events in the trace.

Example 4.3 (Performance profile). Let us consider the event log in Table 1. By considering the timestamp information of each event, each trace will be translated into a vector of six feature values, one for each performance, defined as follows:

		in hours				
			Time Difference			
TraceId	Size	Duration	Min	Max	Mean	Median
1	5	195.3	20.10	125.1	48.84	23.07
2	5	216.5	0.67	141.1	54.14	2.07

The *Transition profile* is defined from the control-flow perspective, that focuses on the ordering of activities in a trace [1]. Features represent the direct following relations between the activities of the trace. For any combination of two activity names, e.g. (A, B), a transition feature is defined (A $\rightarrow$ B) that counts how many times an event with the activity named A has been directly followed by another event with the activity named B in the trace.

Example 4.4 (Transition profile). Let us consider the event log in Table 1. It contains eight distinct transition relations, namely Register Request $\rightarrow$ Examine thoroughly (R $\rightarrow$ ET), Examine thoroughly $\rightarrow$ Check ticket (ER $\rightarrow$ CT), Check ticket $\rightarrow$ Decide (CT $\rightarrow$ D), Decide $\rightarrow$ Reject request (D $\rightarrow$ RR), Register request $\rightarrow$ Check ticket (R $\rightarrow$ CT), Check ticket $\rightarrow$ Examine casually (CT $\rightarrow$ EC), Examine casually $\rightarrow$ Decide (EC $\rightarrow$ D) and Decide $\rightarrow$ Pay compensation (D $\rightarrow$ PC). Each trace will be translated into a vector of eight feature values, one for each distinct transition relation. These features measure the number of times a specific transition occurs in the trace. They are defined as follows:

TraceId	R → ET	ET → CT	CT → D	...	D → PC
1	1	1	1	...	0
2	0	0	0	...	1

4.2 Co-Training Strategy

In keeping with the co-training idea expressed in [18], we illustrate an algorithm for iteratively modifying the similarity matrix associated with a trace profile by using the trace clustering models computed for every other profile. This is done by resorting to a matrix representation of the trace clustering pattern computed for every other profile and using it to update the similarity matrix of the profile considered. Before presenting the algorithm, we introduce the concepts of the similarity matrix and the clustering matrix. Definitions reported in the followings are formulated by assuming that $\mathcal{L}$ denotes an event log recording n traces of a process model, $\mathcal{P}$ denotes a set of trace profiles and $P \in \mathcal{P}$ denotes a given trace profile.

Definition 4.1 (Similarity matrix). *The similarity matrix $\mathcal{S}^P$, associated with event log $\mathcal{L}$ and profile P, is a matrix $(n \times n)$ having a row i (column j) for each trace $T_i \in \mathcal{L}$ ($T_j \in \mathcal{L}$). It is initialized so that $s^P(i,j) = 1 - distance(P, T_i, T_j)$, where $distance(P, T_i, T_j)$ is the distance computed between the vectors of standardized*[1] *features that are computed for both T_i and T_j in the profile P. The distance is computed in the range $[0, 1]$.*

1. Each feature of a vector space model is normalized in the interval [0,1] at the beginning of the matrix computation, in order to give equal importance to all the features in the vector space.

Definition 4.2 (Clustering matrix). *Let $\mathcal{S}^P$ be the similarity matrix associated with event log $\mathcal{L}$ and profile P and $\mathcal{C}^P$ be a clustering pattern that groups the traces of $\mathcal{L}$ into k trace clusters, according to the similarities loaded in $\mathcal{S}^P$. The clustering matrix $\mathcal{C}^P$ is a matrix $(n \times k)$ having a row i for each trace $T_i \in \mathcal{L}$ and a column j for each cluster $C_j \in \mathcal{C}^P$, such that:*

$$c^P(i,j) = \begin{cases} \sqrt{\frac{1}{size(C_j)}} & \text{if } T_i \in C_j, \\ 0 & \text{otherwise}, \end{cases} \tag{1}$$

where $size(C_j)$ is the number of traces of $\mathcal{L}$, which are clustered in C_j. According to this choice, $\mathcal{C}^P \times tran(\mathcal{C}^P)$ is a matrix whose rows sum to the unit, where $tran(\cdot)$ determines the transpose of a matrix.

The co-training strategy requires the similarity matrix of a profile to be updated on the basis of the clustering matrices of the other profiles.

Definition 4.3 (Similarity matrix co-training). *Let $\mathcal{S}^P$ be the similarity matrix associated with an event log $\mathcal{L}$ and a profile P, and $\wp(\mathcal{C}, \neg P)$ the set of clustering matrices $\mathcal{C}^Q$ associated with $\mathcal{L}$ and yielded for every other profile $Q \in \mathcal{P}$ with $Q \neq P$ (i.e. $\wp(\mathcal{C}, \neg P) = \{\mathcal{C}^Q | Q \in \mathcal{P}, Q \neq P\}$). The similarity matrix $\mathcal{S}^P_{new}$ is computed by projecting $\mathcal{S}^P$ onto the sum of the clustering matrices in $\wp(\mathcal{C}, \neq P)$. Formally:*

$$\mathcal{S}^P_{new} = sym\left(\mathcal{S}^P \times \sum_{\mathcal{C}^Q \in \wp(\mathcal{C}, \neq P)} \left(\mathcal{C}^Q \times tran(\mathcal{C}^Q)\right)\right), \tag{2}$$

where $sym(\cdot)$ computes a symmetrization of a matrix (with $sym(S) = (S + tran(S))/2$). Since the projection matrix is orthogonal, the inverse projection is done using its transpose. Finally, the symmetrization step is performed here since the projection operator may not yield a symmetric matrix.

It is noteworthy that, in Formula 2, new similarities for a selected profile are obtained by averaging out the similarities within clusterings associated with every other profile. The across cluster similarities are also averaged out in the new similarity matrix. This implies that the projection in the space of clustering matrices makes similarities within a cluster higher, throwing away the intra-cluster information that is irrelevant for clustering. The projection operator reported in Formula 2 is mainly inspired by the operator that Kumar and Daumé [18] defined for spectral clustering with co-training. In particular, the original spectral projection operator makes a projection of a graph structure on the eigenvectors of the other graph Laplacian space, while our projection operator makes a projection of one similarity matrix on the clustering matrices of every other trace profile. Kumar and Daumé [18] observe that, as the number of iterations increases, the use of their spectral projection operator in the subspace of discriminative eigenvectors makes the edges within a spectral cluster closer to each other. Similarly, we may expect that the similarities within a cluster diffuse from one to the other when projecting one similarity matrix on every other clustering matrix.

The across cluster similarities also diffuse to one another. Finally, we point out that, according to Definition 4.2, all traces in the same cluster are treated in a similar way to and differently from traces in other clusters in Formula 2. Therefore, iterating the projection process should fix the traces in a cluster, such that they will tend to be clustered together when the modified similarity matrix is used to recompute the subsequent clustering pattern in the next iteration.

Algorithm 1. CoTraDiC($\mathcal{L}, \mathcal{P}$) Return $\mathcal{C}$

```
Require: $\mathcal{L}$ {Event log} $\mathcal{P}$ {Trace profile set} $MaxIt$ {Maximum number of iterations} $\theta$ {Clustering parameters}
Ensure: $\mathcal{C}$ {Traces clustered}
 1: $it \leftarrow 0$ $\mathcal{P}(\mathcal{C})_{it} \leftarrow \emptyset$
 2: for $P \in \mathcal{P}$ do
 3:     $\mathcal{L}^P \leftarrow$profile($\mathcal{L}, P$)
 4:     $\mathcal{S}^P_{it} \leftarrow$similarityMatrix($\mathcal{L}^P$)
 5:     $\mathcal{C}^P_{it} \leftarrow$ clustering($\mathcal{S}^P_{it}, \theta$)
 6:     $\mathcal{P}(\mathcal{C})_{it} \leftarrow \mathcal{P}(\mathcal{C})_{it} \cup \{\mathcal{C}^P_{it}\}$
 7: end for
 8: repeat
 9:     $it \leftarrow it + 1$
10:     $\mathcal{P}(\mathcal{C})_{it} \leftarrow \emptyset$
11:     for $P \in \mathcal{P}$ do
12:         $\mathcal{S}^P_{it} \leftarrow \mathcal{S}^P_{it-1} \sum_{Q \in \mathcal{P}, Q \neq P} (\mathcal{C}^Q_{it-1} \times \mathrm{transpose}(\mathcal{C}^Q_{it-1}))$
13:         $\mathcal{S}^P_{it} \leftarrow$symmetrize($\mathcal{S}^P_{it}$)
14:         $\mathcal{C}^P_{it} \leftarrow$ clustering($\mathcal{S}^P_{it}, \theta$)
15:         $\mathcal{P}(\mathcal{C})_{it} \leftarrow \mathcal{P}(\mathcal{C})_{it} \cup \{\mathcal{C}^P_{it}\}$
16:     end for
17: until silhouette(consensus($\mathcal{P}(\mathcal{C})_{it}, \theta$)) $<$ silhouette(consensus($\mathcal{P}(\mathcal{C})_{it-1}, \theta$)) OR $it \geq MaxIt$
18: if silhouette(consensus($\mathcal{P}(\mathcal{C})_{it}, \theta$)) $\geq$ silhouette(consensus($\mathcal{P}(\mathcal{C})_{it-1}, \theta$)) then
19:     $\mathcal{C} \leftarrow$consensus($\mathcal{P}(\mathcal{C})_{it}, \theta$)
20: else
21:     $\mathcal{C} \leftarrow$consensus($\mathcal{P}(\mathcal{C})_{it-1}, \theta$)
22: end if
```

The learning algorithm, reported in Algorithm 1, is three-stepped. It consists of an initialization step (lines 1-7, Algorithm 1), an iterative co-training step (lines 8-17, Algorithm 1) and a consensus clustering step (lines 18-22, Algorithm 1). Input parameters of the algorithm are the event log $\mathcal{L}$, the set $\mathcal{P}$ of multiple trace profiles, the maximum number of iterations $MaxIt$ and the parameters θ requested by the clustering algorithm (e.g. number of clusters k with k-partitioning algorithms). The output is a trace clustering pattern $\mathcal{C}$, yielded by clustering the traces of $\mathcal{L}$ across the multiple profiles of $\mathcal{P}$.

We begin by describing the initialization step that starts by assigning zero to the iterator it of the iterative co-training step. For each profile $P \in \mathcal{P}$, we determine: (a) the initial similarity matrix (i.e. $\mathcal{S}^P_{it}$ at line 4 of Algorithm 1) of the traces in $\mathcal{L}$, computed with the vector space model of the traces, which is associated with selected profile P (see Definition 4.1) and (b) the initial clustering matrix (i.e. $\mathcal{C}^P_{it}$ at line 5 of Algorithm 1) of the traces in $\mathcal{L}$, computed by applying a distance-based clustering algorithm with the similarity matrix associated with profile P (see Definition 4.2).

For each trace profile, we initialize the similarity values for the traces of the event log by computing a vector space distance (e.g. the euclidean distance) between each pair of traces (line 4, Algorithm 1). The distance metric is computed by comparing the feature values of the traces, which are built for the profile considered. In the iterative co-training step (lines 8-17, Algorithm 1), we use every clustering matrix, within the iterative co-training strategy, in order to determine both the new similarity matrix (lines 12-13, Algorithm 1) and the new clustering matrix (line 14, Algorithm 1) of every other trace profile (see Definition 4.3). In particular, for a certain profile $P \in \mathcal{P}$, this iterative process bases the computation of the new similarity matrix associated with P, at the present iteration it (i.e. $\mathcal{S}^P_{it}$ at lines 12-13 Algorithm 1), on the similarity matrix associated with P, at the previous iteration $it - 1$ (i.e. $\mathcal{S}^P_{it-1}$ at line 12 of Algorithm 1), as well as on the clustering matrices associated with every other profile $Q \in \mathcal{P}$ with $Q \neq P$, at the previous iteration $it - 1$ (i.e. $\mathcal{C}^Q_{it-1}$ at line 12 of Algorithm 1). The updated similarity matrix of every profile (i.e. $\mathcal{S}^P_{it}$ at line 14, Algorithm 1) is then used to recompute the clustering matrix associated with the same profile (i.e. $\mathcal{C}^P_{it}$ at line 14, Algorithm 1). We stop the iterative co-training process when the compactness of the consensus clustering pattern (see Details in Section 4.3), computed from the clustering matrices yielded in the last iteration it, is less than the compactness of the consensus clustering pattern yielded in the penultimate iteration $it - 1$ (line 17, Algorithm 1). The underlying motive for this stopping criterion is to favor clusters that are compact. Compact clusters have a lot of significance in pattern classification, where the objective is to enable the discovery of decision boundaries. We use the Silhouette width [31] to measure the compactness of each consensus clustering pattern. It is a succinct representation of how well each trace lies within its cluster. It assumes values in the range $[-1, 1]$. A value of the Silhouette width closer to one means that the trace is appropriately clustered. We calculate the Silhouette width of the consensus clustering pattern with respect to each trace profile considered and determine the mean of the Silhouette widths computed for all the profiles. As a further stopping criterion, we stop the iterative process when the co-training updates are performed for a fixed number ($MaxIt$) of iterations (line 17, Algorithm 1).

Finally, in the consensus clustering step, we map the multiple clustering matrices, which are iteratively computed with the co-training strategy from the multiple profiles, to a single consensus clustering pattern. It is computed by combining the clustering matrices that are updated with the co-training step and have the highest average compactness on the multiple profiles considered (lines 19, 21, Algorithm 1). We describe the procedure to compute this consensus clustering pattern in the next section.

4.3 Consensus Clustering Pattern

The consensus clustering pattern is computed by adapting the merge strategy of the split-and-merge ensemble strategy [29] to the task in hand (see Algorithm 2). Let $\mathcal{P}(\mathcal{C})$ be the set of trace clustering patterns (or equivalently clustering matrices) computed with the co-training strategy for the trace profiles of this study. We recall that each clustering matrix is

$(n \times k)$ with rows representing the traces of the event log processed and columns representing the clusters computed in the specific profile. Therefore, we can consider each row of a clustering matrix as a vector space model of the associated trace, compute the similarities between these rows and use these similarity values to populate a consensus similarity matrix of the event log $\mathcal{L}$ (lines 1-4, Alg. 2). Finally, we can compute the final consensus clustering pattern from this consensus similarity matrix (line 5, Algorithm 2).

Algorithm 2. Consensus($\mathcal{P}(\mathcal{C}), \theta$) Return $\mathcal{C}$

```
Require: $\mathcal{P}(\mathcal{C})$ {a set of clustering patterns of the event log $\mathcal{L}$, one computed for each trace profile} $\theta$ {Clustering parameters}
Ensure: $\mathcal{C}$ {the traces of $\mathcal{L}$ clustered according to a consensus clustering pattern computed}
1: for $\mathcal{C} \in \mathcal{P}(\mathcal{C})$ do
2:     $\mathcal{S}^{\mathcal{C}}$ ←similarityMatrix(rowSet($\mathcal{C}$)) {similarity matrix computed between each pair of rows of $\mathcal{C}$}
3: end for
4: $\mathcal{S} \leftarrow \frac{1}{\text{cardinality}(\mathcal{P}(\mathcal{C}))} \sum_{\mathcal{C} \in \mathcal{P}(\mathcal{C})} \mathcal{S}^{\mathcal{C}}$
5: $\mathcal{C}$ ←clustering($\mathcal{S}, \theta$)
```

Procedurally, we begin by considering the clustering matrices, updated for each trace profile, and computing a similarity matrix for the rows stored in every clustering matrix under analysis (lines 1-3, Algorithm 2).

Definition 4.4 (Similarity matrix for the rows of a clustering matrix). *Let $\mathcal{C}^P(n \times k)$ be a clustering matrix of the event log $\mathcal{L}$, which is updated with the co-training step for the trace profile $P \in \mathcal{P}$. The similarity matrix $\mathcal{S}^{\mathcal{C}^P}(n \times n)$ collects the similarities computed between each pair of rows in $\mathcal{C}^P$, such that: $\mathcal{S}^{\mathcal{C}^P}(i, j) = 1 - distance(\mathbf{c}_i^P, \mathbf{c}_j^P)$, where $\mathbf{c}_i^P$ and $\mathbf{c}_j^P$ denote the row vector i and the row vector j of $\mathcal{C}^P$, respectively (see Definition 4.2) and $distance(\mathbf{c}_i^P, \mathbf{c}_j^P)$ denotes the distance metric selected and computed between these two row vectors.*

Then we compute the consensus similarity matrix by summing the similarity matrices of the rows of the clustering matrices, which are computed for the multiple trace profiles (see Definition 4.4), and normalizing the result with respect to the number of matrices summed (line 4, Algorithm 2).

Definition 4.5 (Consensus similarity matrix). *Let $\mathcal{L}$ be the event log recording n traces, $\mathcal{P}(\mathcal{S}^P(n \times k))$ be a set of the similarity matrices computed for the event log $\mathcal{L}$. The consensus similarity matrix $\mathcal{S}(n \times n)$ is computed as follows: $s(i,j) = \frac{\sum_{\mathcal{S}^P(n \times k) \in \mathcal{P}(\mathcal{S}^P(n \times k))} \mathcal{S}^{\mathcal{S}^P}(i,j)}{cardinality(\mathcal{P}(\mathcal{S}^P(n \times k)))}$, where $cardinality(\mathcal{P}(\mathcal{S}^P(n \times k)))$ is the number of patterns processed for the consensus (or equivalently the number of trace profiles).*

Finally, we use the distance-based clustering algorithm selected, in order to determine a consensus pattern $\mathcal{C}$ that represents the traces of $\mathcal{L}$ now clustered according to the consensus similarity matrix $\mathcal{S}$ (line 5, Algorithm 2).

5 Time Complexity Analysis

For this analysis we assume that n denotes the number of traces in the event log, m the number of trace profiles in the co-training strategy, M the average number of features per trace profile and k the average number of clusters per run. In addition, we assume that the time cost of each distance-based clustering algorithm selected can be denoted with O (d) in this analysis. The time complexity of CoTraDiC depends on the cost of initializing one similarity matrix and one clustering matrix of the event log $\mathcal{L}$ for each trace profile, the cost of performing the co-training updates of both the similarity matrix and the clustering matrix of each profile, as well as the cost of computing the consensus clustering pattern and measuring its compactness at each co-training update. The time cost of computing the similarity matrix of the event log for a specific trace profile is O(n^2M), while the cost of clustering the traces according to a profile similarity matrix is assumed to be O(d). The time cost of performing the co-training updates is here calculated by considering the matrix multiplication as the leading operation of the updates. We note that Formula 2, that is used to perform the co-training updates, comprises $m-2$ multiplications with square matrices $(n \times n)$ and $m-1$ multiplications with rectangular matrices ($(n \times k)$ and $(k \times n)$). Thus, the time cost of computing this Formula, for a specific profile, is O ($(m-2)n^{2.376} + (m-1)n^2k$), with O($n^{2.376}$) the time cost of multiplying square matrices $(n \times n)$ by resorting to the CoppersmithWinograd algorithm [32] and O(n^2k) the time cost of multiplying rectangular matrices $(n \times k)$ and $(k \times n)$. The time cost of computing a consensus clustering pattern depends on the cost of computing the consensus similarity matrix and the cost of performing a distance-based clustering by using this similarity matrix. The computation of a consensus similarity matrix has a time cost O(mn^2k), that is, the cost O(n^2k) of computing a clustering matrix rows similarity matrix for each one of the m trace clustering patterns processed for the consensus. Thus, the total time cost is O ($mn^2k + d$). The cost of computing the Silhouette width of the consensually clustered traces is O(n^2). As we compute the Silhouette width with respect to each trace profile, the time cost of measuring the compactness of the consensus clustering pattern is O(mn^2). Both the consensus pattern and the Silhouette width are computed at each iteration of the co-training phase, therefore this cost is multiplied by the number of performed iterations. Globally the time cost of completing the clustering process is $(m \underbrace{n^2M}_{initial\ \mathcal{S}} + m \underbrace{d}_{initial\ \mathcal{C}} + MaxIt \cdot (m \underbrace{((m-2)n^{2.376} + (m-1)n^2k)}_{co\text{-}trained\ \mathcal{S}} + m \underbrace{d}_{co\text{-}trained\ \mathcal{C}} + \underbrace{mn^2k + d}_{consensus} + \underbrace{mn^2}_{compactness}))$, in the worst case, that is, the maximum number of iterations ($MaxIt$) is completed. This global cost is equal to $\mathbf{mn^2}(\mathbf{M} + \mathbf{mkMaxIt} + MaxIt + (\mathbf{m-2})\mathbf{n^{0.376}MaxIt}) + \mathbf{d}(m + \mathbf{mMaxIt} + MaxIt)$, that is, $(mn^2(M + mkMaxIt + (m-2)n^{0.376}MaxIt) + dmMaxIt)$, by considering the bold parts of the formula, which are asymptotically the most complex in the formula. Generally $M < (mkMaxIt + (m-2)n^{0.376}MaxIt) < (mkMaxIt + mn^{0.376}MaxIt)$, so the time complexity of the algorithm is $O(m^2n^2MaxIt(k + n^{0.376}) + dmMaxIt)$.

6 Experiments

The multiple view trace clustering algorithm presented in this paper, whose implementation is available to the public, is written in Java.[2] It is evaluated on several benchmark event logs by considering both machine learning and process mining metrics. The evaluation study aims to seek answers to the following questions: (1) Can trace clusters, discovered with the co-training strategy, be compact clusters tightly grouping well separated traces when evaluated by considering the several perspectives of process mining? (2) Are trace clusters discovered with co-training more compact than clusters discovered by using competitor multi-view learning strategies? (3) Are trace clusters able to minimize the difficulty of modeling spaghetti-like processes by allowing us to discover good process models from the logs (i.e. simple models which fit traces at the best)? (4) Are process models discovered through traces clustered with the co-training strategy better if compared to the process models constructed after clustering traces by using either a traditional clustering algorithm with a single process mining perspective or a competitor (e.g. ensemble) multi-view learning strategy with all perspectives? Experiments are run on an Intel(R) Core(TM) i7-2670QM CPU@2.20 GHz running Windows 7 Professional.

6.1 Event Logs

We use 11 event logs exhibiting a variety of process behaviors. All are taken from the process mining repository.[3] Event logs contain information on activities, resources and time of traces. The only exception is ETM, that is the only event log used in this study without information on resources. The event logs *cProblem* (1,487 traces, 6,660 events), *incident* (7,554 traces, 65,533 events) and *oProblem* (819 traces, 351 events) are real-life event logs from Volvo IT Belgium. They contain events from an incident and problem management system called VINST. In particular, cProblem collects traces for closed problems, while oProblem collects traces for open problems only. The event log incident collects traces for mixed problems. These three event logs are part of the BPI Challenge 2013 [33]. The event log *etm* (475 traces, 2,440 events) is an artificial log describing a loan application process. The event log *hospital* (1,143 traces, 150,291 events) is a real-life log of a Gynecology department of a Dutch academic hospital. It was originally intended for use in the BPI Challenge 2011 [34]. The event log *isbpm* (2,000 traces, 28,534 events) is an artificial log with both noise and increasing trace length. It is used for decomposed conformance checking purposes. The event log *photo* (100 traces, 40,995 events) is an artificial log about a digital photo copier. It is introduced in [35] for evaluating process modeling techniques. The event log *repair* (1,104 traces, 11,855 events) is about a process to repair telephones in a company. It is used in the process mining book [1]. The event logs *review* (100 traces, 3,730 events) and *lReview* (10,000 traces, 236,360 events) are logs, both handling reviews for a journal. Finally, the event log *claims* (3,512 traces, 46,138 events) describes the handling of claims in an insurance company. Both review and claims are used in [1].

6.2 Experimental Set-Up

Our study aims at evaluating the effectiveness of the co-training strategy when it is applied to cluster a multiple profile representation of the traces of a generic process.

6.2.1 Evaluation Metrics

We evaluate the significance of the clusters discovered by resorting to a machine learning perspective, as well as to a process mining perspective.

For the *machine learning perspective* , we compute the *Silhouette width* [31], that is a machine learning metric, in order to analyze the compactness of a clustering pattern. We calculate the Silhouette width of the clustering pattern with respect to each trace profile of the event log. As the objective of multiple view learning is to compute a single clustering pattern that is compact enough with respect to the several trace profiles, we analyze the mean of the Silhouette width computed per profile (*average multiple view Silhouette width*). Additionally, we consider the time (in milliseconds) spent to complete the clustering process and determine clusters of traces, as well as the number of iterations performed to complete the learning process with the co-training strategy.

For the *process mining perspective* , the objective of clustering traces of an event log is to ease the discovery of process models, by grouping together traces that conform to similar execution patterns/behavior. Thus, we evaluate the significance of the clusters formed by evaluating the process models, which can be discovered from the traces within each cluster. As discussed in [6], clusters of traces are meaningful when all traces belonging to related cases are in the same cluster and traces that are unrelated are not. When clusters are meaningful, process models resulting from traces in each cluster are less complex (more comprehensible and less spaghetti-like) and have a high degree of fitness.[4] In this study, we generate process models using the *Alpha*++ mining algorithm [36] (available in the ProM framework). Therefore, the discovered process models are expressed in terms of Petri nets, which allow for the modeling of concurrency by a bipartite graph consisting of places and transitions. An example of a Petri net is shown in Fig. 1. We compute one process model from the entire log of traces, as well as one process model from each cluster of traces. We use the Petri-Net Complexity Analysis plugin in ProM, in order to measure the complexity of the process models thus discovered. The complexity analysis plugin generates metrics such as the *number of control-flows* , *and-joins* , *and-splits* , *xor-joins* , *xor-splits* , *arcs* , *places* and *transitions* (see Fig. 1) in the process model. The larger the value of these metrics, the more complex the model [37]. Finally, we use the conformance checker plugin in ProM, in order to relate events in the event log to activities in the process model and compare both [1]. We analyze the *fitness* metric, in order to evaluate the conformance checking of a process model. Fitness evaluates the proportion of behavior, seen with traces in the event log, possible with the discovered model. It assumes values in the range $[0,1]$. In particular, fitness is computed by replaying a trace from the model and recording all situations where a transition is forced to fire without being enabled, i.e. we count the missing tokens. Formally, $fitness(t,P) = \frac{1}{2}(1-\frac{mt}{c}) + \frac{1}{2}(1-\frac{r}{p})$ (chapter 7, p. 198 [1]), where t is the trace to be replayed according to the process

2. http://www.di.uniba.it/ appice/software/CoTraDiC/index.htm
3. http://www.processmining.org/logs/start

4. The analysis of how deadlocks, live-locks, lack of synchronization and and sub-processes are supported by Petri-nets is out of the scope of this study.

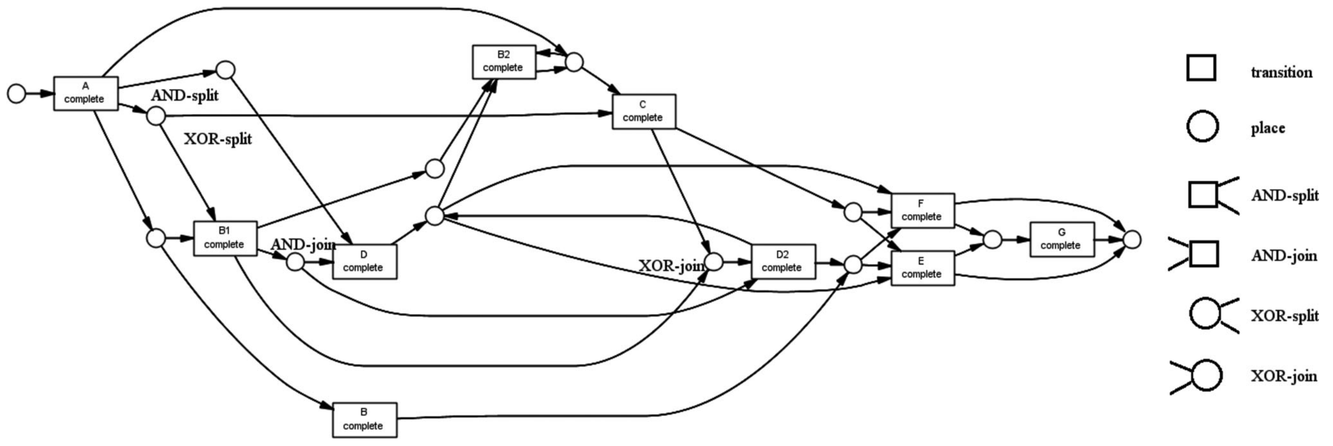


Fig. 1. The Petri net (consisting of places, transitions, and arcs) computed from Alpha++ by processing all traces (without clustering) of the log etm. Arcs run from a place to a transition or vice-versa, never between places or between transitions. Control-flows run from a transition to a transition.

model P, mt is the number of missing tokens, c is the number of consumed tokens, r is the number of remaining tokens and p is the number of produced tokens. Hence, if the log can be replayed correctly, i.e., there are tokens neither missing nor remaining when replaying the log on the Petri net, fitness equals 1. In the worst case, every produced and consumed token is remaining or missing, fitness equals 0. As suggested in [6], we calculate the weighted average of these metrics, in order to balance metrics over possible unbalanced clusters. In particular, for each complexity and conformance metric, we calculate: $m_{Wavg} = \sum_{C \in \mathcal{C}} (size(C) \times m(C)) / \sum_{C \in \mathcal{C}} size(C)$, where m is the metric, $\mathcal{C}$ is the clustering pattern of an event log, while $size(C)$ is the number of traces falling in a cluster $C \in \mathcal{C}$.

6.2.2 Compared Algorithms

We use the k-medoids algorithm [38] as a base distance-based algorithm and the euclidean measure as a base distance metric. Both are commonly used in the process mining practice. The k-medoids algorithm allows us to control the number of clusters to be discovered[5] without requiring any additional input parameters. Similarly to k-means, it generates a partition of the data such that objects in a cluster are more similar to each other than they are to objects in other clusters. However, it is less sensitive than k-means to outliers, where an extremely large value may substantially distort the distribution of data.[6] On the other hand, the k-medoids algorithm is much more expensive. Indeed, it involves computing all pairwise distances, that is O(n^2), whereas k-means runs in O(kn). The euclidean distance is simple to compute on a feature-vector representation of data. It is a proper choice in a continuous space, where all dimensions are properly scaled and relevant. In any case, the learning strategy presented here can be used with any distance-based algorithm, as well as with any feature-vector distance measure. We determine the clustering pattern by accounting for multiple trace profiles with the co-training strategy (*coTraDiC*), as described in this paper. We set the maximum number of iterations to fifteen ($MaxIt = 15$).[7] We compare clustering patterns discovered by the co-training strategy with clustering patterns discovered by multiple view competitor strategies. As multiple view competitors we consider: (1) Multiple view clustering patterns, which are discovered by concatenating the features of each trace profile and running the k-medoid algorithm with the euclidean distance on this joint view representation of the traces (*jCLUS*). (2) Multiple view clustering patterns, which are discovered according to the split-and-merge ensemble strategy [29], by running repeatedly the k-medoid algorithm with the euclidean distance on the separate trace profiles, and then combining single-view originated clustering patterns in a single pattern (*ensambleCLUS*). The selected ensemble strategy is that used to combine cluster final results of a co-training process (see Algorithm 2). Thus, this represents the baseline of the co-training strategy, when no iteration is really performed.

We run five trials of algorithms which are compared in this study. For the machine learning evaluation, we calculate the mean of the average multiple view Silhouette width of the clustering patterns, computed with these trials, and repeat this analysis with the number of cluster k ranging between 2, 3, 4 and 5. For each k value, we use the one-way analysis of variance (ANOVA) [39], in order to determine whether there are any significant differences between the means of the average multiple view Silhouette width of the three compared algorithms on the tested trials. We test the hypothesis that they are all the same against the general alternative that they are not all the same. If this hypothesis is rejected, we run a multiple comparison test, in order to determine whether any of those means are significantly different from each other [39]. The significance level of this statistical analysis is 0.05.[8] For the process model evaluation, the clustering pattern of the trial maximizing the average multiple view Silhouette width is that used for the

5. The presented system also allows us to specify a different number of clusters for each profile.

6. While k-means takes the mean value of the objects in a cluster as a reference point, the k-medoids uses the most centrally located object in a cluster, which can be assimilated to the median.

7. The results reported in Table 4 show that, in all the logs considered in this study, the co-training phase is completed in less than five iterations.

8. We use the Matlab functions, *anova1*(...) and *multcompare*(...), in order to perform the one-way analysis of variance and the multiple comparison test, respectively.

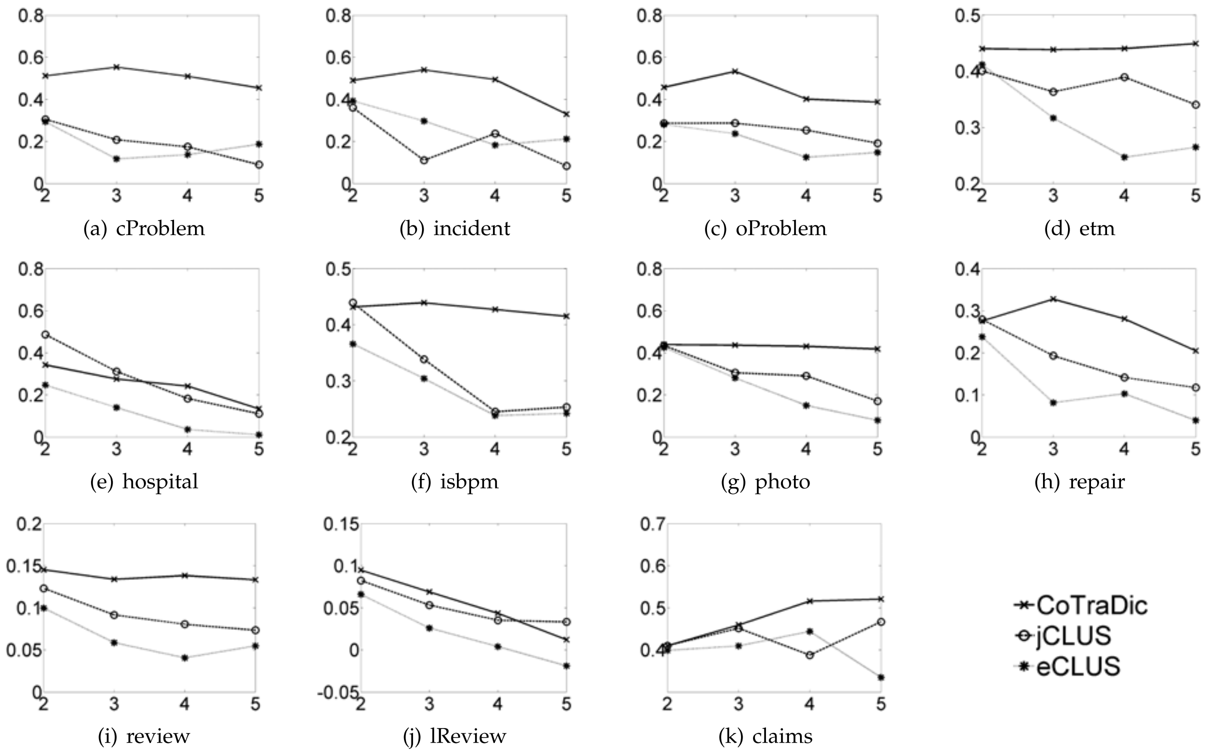


Fig. 2. Average multiple view Silhouette width analysis: mean of the average multiple view Silhouette width (Y-axis) computed on five trials run per algorithm (CoTraDiC, jCLUS, and eCLUS) with number of cluster k ranging between 2, 3, 4, and 5 (X-axis).

cluster-based process model discovery. Process models are discovered with $k = 2$, as we intend to achieve the objective of simplifying the process model complexity, without proliferating models per event log. For this analysis, we investigate the viability of a multiple view learning strategy in process mining, by also comparing process models, computed with a multiple view clustering strategy (co-training, joint-view or ensemble), to process models computed with a single view strategy. For the single-view learning, we compute a trace clustering pattern by running the standard k-medoid algorithm with the euclidean distance on a specific trace profile of the event log, as described in [5].

6.3 Results and Discussion

The analysis of the results is illustrated in the following.

6.3.1 *Silhouette Width Analysis*

Figs. 2a, 2b, 2c, 2d, 2e, 2f, 2g, 2h, 2i, 2j, and 2k report the measures of compactness (vertical axis) for the clustering patterns discovered by both CoTraDiC, jCLUS and eCLUS, by varying the number of cluster k between 2, 3, 4 and 5 (horizontal axis). Compactness is measured by the average multiple view Silhouette width (details in Section 6.2.1). Table 2 collects the results of the one-way analysis of the variance of compactness and the multiple comparison test between CoTraDiC, jCLUS and eCLUS. The results confirm that the clustering patterns discovered by CoTraDiC are more compact than the clustering patterns discovered by its multiple view competitors (jCLUS and eCLUS). The statistical analysis shows that this behavior is robust for the initial medoid choice. Results in Table 2 show that the mean of the compactness of clusterings computed by CoTraDiC is generally significantly better than the mean computed for its competitors. Finally, this analysis, repeated by varying the number of cluster k, reveals that our conclusions are robust for variations in k.

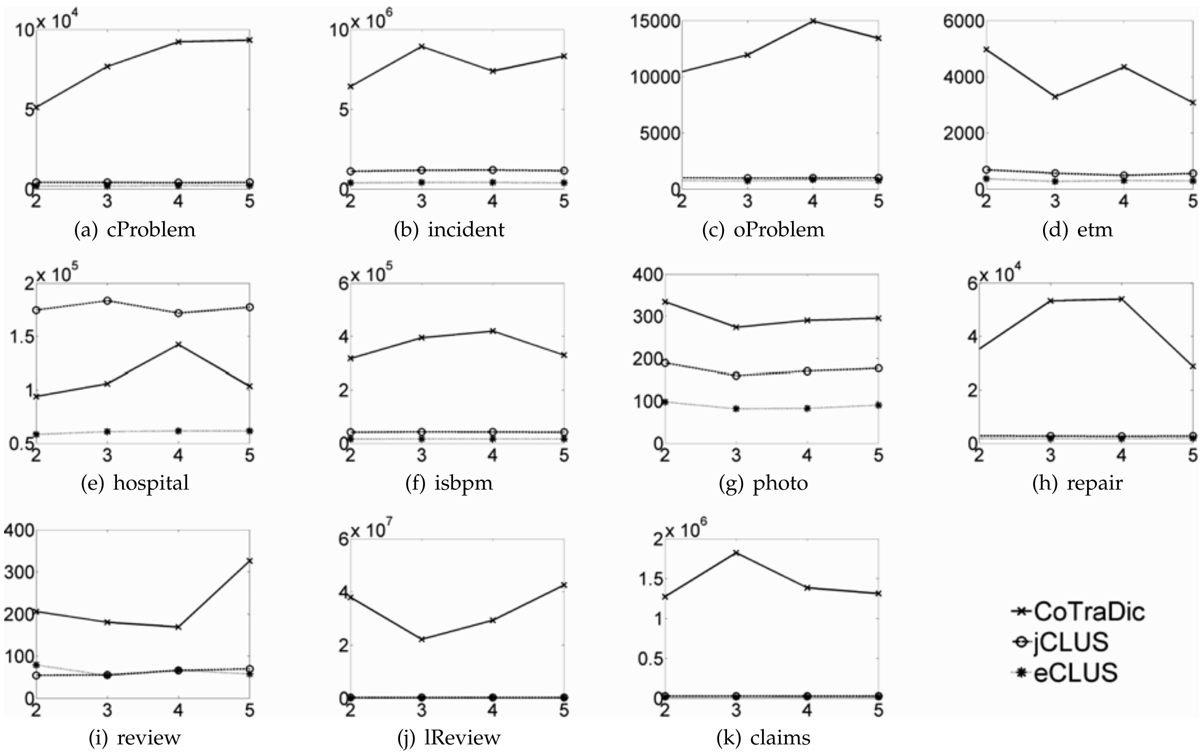
6.3.2 *Learning Time Analysis*

Figs. 3a, 3b, 3c, 3d, 3e, 3f, 3g, 3h, 3i, 3j, and 3k report the computation time (in millisecs, vertical axis) spent by both CoTraDiC, jCLUS and eCLUS to complete the multiple view clustering process, by varying the number of cluster k between 2, 3, 4 and 5 (horizontal axis). As suggested by the

TABLE 2
Silhouette Width Analysis: Results of the One-Way Analysis of Variance and the Multiple Comparison Test between CoTraDiC, jCLUS, and eCLUS

log/k	CoTraDiC vs jCLUS				CoTraDiC vs eCLUS			
	2	3	4	5	2	3	4	5
cProblem	+	+	+	+	+	+	+	+
incident	+	+	+	=	+	+	+	=
oProblem	+	+	+	+	+	+	+	+
etm	=	=	=	=	=	=	=	=
hospital	=	=	=	=	=	+	+	+
isbpm	=	+	+	+	=	+	+	+
photo	=	+	+	+	+	+	+	+
repair	=	+	+	+	=	+	+	+
review	=	+	+	+	=	+	+	+
lReview	=	=	=	=	+	+	=	=
claims	=	=	=	=	=	=	=	=

We report + (-) when CoTraDiC has the mean of the average multiple view Silhouette width that is significantly higher (lower) than the considered competitor with significance value 0.05.



(a) cProblem (b) incident (c) oProblem (d) etm
(e) hospital (f) isbpm (g) photo (h) repair
(i) review (j) lReview (k) claims

Fig. 3. Computation time (in milliseconds) analysis: mean computed on five trials (Y-axis), run per algorithm (CoTraDiC, jCLUS, and eCLUS), with k ranging between 2, 3, 4, and 5 (X-axis).

TABLE 3
Time Complexity Analysis (CoTraDic, jCLUS, eCLUS): n Denotes the Number of Traces in the Event Log, m the Number of Trace Profiles in the Co-Training Strategy, M the Average Number of Features per Trace Profile, k the Number of Clusters per run, and d the Time Cost of the Base Distance-Based Clustering Algorithm

		Time complexity
CoTraDiC	see Section 5	$O(m^2n^2MaxIt(k+n^{0.376})+dmMaxIt)$
jCLUS	$\underbrace{n^2mM}_{joint\ distance\ matrix} + \underbrace{d}_{joint\ clustering}$	$O(mn^2M+d)$
eCLUS	$m \underbrace{(n^2M+d)}_{single\ view\ clustering} + \underbrace{mn^2k+d}_{consensus}$	$O(mn^2M+mn^2k+md)$

If k-medoid is run as the base clustering algorithm, then $d = n(n-k)NIter)$, where $NIter$ is the number of iterations during k-medoid.

asymptotic analysis of the time complexity of CoTraDic, as well as of its multi-view competitors jCLUS and eCLUS (see Table 3), the iterative learning strategy used with co-training slows down the clustering process. In any case, the number of iterations, performed in the co-training phase and reported in Table 4, shows that CoTraDiC converges in a small number of iterations (less than 5), never stopping with the maximum number of iteration criterion ($MaxIt$). This confirms that the compactness-based criterion, that we have defined for this task, is an efficacious stopping criterion that guarantees the convergence of the co-training phase in few iterations.

6.3.3 Process Model Analysis

Table 5 collects metrics which measure both complexity and conformance (details in Section 6.2.1) of Petri nets computed from the traces of the entire event log, as well as from the clustered traces. In this analysis, we evaluate the clustering patterns discovered by both multiple view algorithms (CoTraDiC, jCLUS and eCLUS) and single-view algorithms (activity—aCLUS, resource—rCLUS, performance—pCLUS and transition—tCLUS).

TABLE 4
Number of Iterations Analysis: Average Number of Iterations Performed to Complete the Learning Phase

log/k	2	3	4	5	log/k	2	3	4	5
cProblem	1.2	1.8	2.2	2.2	photo	3.2	3.2	3.8	3.6
incident	1.2	1.4	1.6	1.4	repair	2	3	3	1.6
oProblem	1.4	1.6	2	1.8	review	2.5	3	3	5.5
etm	3.4	2.2	3	2.2	lReview	2.8	2.4	2.8	2.2
hospital	1.2	1.4	1.6	1.2	claims	2.2	2.6	2.6	2.4
isbpm	2.4	3.2	3.6	3					

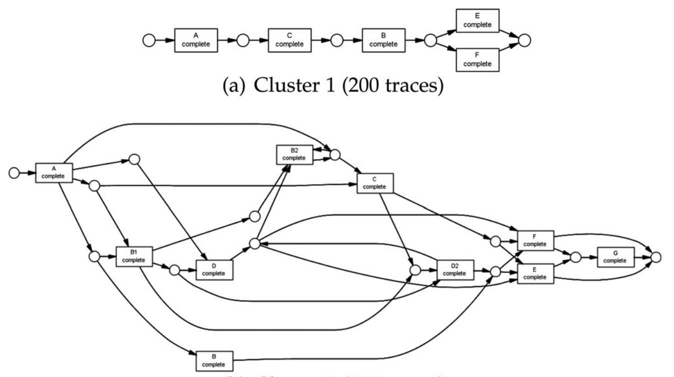
For each log, the average values are computed on five trials, run per CoTraDiC.

TABLE 5
Process Model Analysis: Complexity (Number of Control Flows-CF, Number of AND/OR Splits/Joins, Number of Arcs-A, Number of Places-P and Number of Transitions-T), and Conformance (Fitness-f) of the Computed Petri Nets

log	CF	and/or	A	P	T	f
cProblem	0.00	0.00	5.00	2.00	4.00	0.66
CoTraDiC	0.00	0.00	4.64	2.00	4.00	**0.69***
jCLUS	0.00	0.00	4.75	2.00	3.87	0.67
eCLUS	0.00	0.00	5.00	2.29	4.00	**0.69***
aCLUS	0.00	0.00	4.91	2.00	3.95	*0.68*
rCLUS	0.00	0.00	4.98	2.00	3.99	0.66
pCLUS	0.00	0.00	4.41	2.20	3.20	0.67
tCLUS	0.00	0.00	5.00	2.00	4.00	0.66
incident	0.00	0.00	5.00	2.00	4.00	0.22
CoTraDiC	0.00	0.00	2.85	2.00	3.85	**0.44***
jCLUS	0.00	0.00	4.28	2.00	3.72	0.41
eCLUS	0.00	0.00	4.72	2.00	4.00	0.27
aCLUS	0.00	0.00	4.85	2.85	4.00	0.26
rCLUS	0.00	0.00	5.00	2.00	4.00	0.22
pCLUS	0.00	0.00	4.11	2.00	4.00	*0.27*
tCLUS	0.00	0.00	4.95	2.00	3.98	0.23
oProblem	0.00	0.00	6.00	2.00	3.00	0.52
CoTraDiC	0.00	0.00	5.88	2.00	3.00	0.53
jCLUS	0.00	0.00	5.55	2.00	3.00	0.55
eCLUS	0.00	0.00	5.00	2.29	4.00	**0.69***
aCLUS	0.00	0.00	5.44	2.00	3.00	*0.52*
rCLUS	0.00	0.00	5.99	2.00	3.00	*0.52*
pCLUS	0.00	0.00	5.93	2.00	3.00	*0.52*
tCLUS	0.00	0.00	5.97	2.00	3.00	*0.52*
etm	35.00	25.00	39.00	13.00	10.00	0.81
CoTraDiC	21.53	14.89	26.79	9.63	7.89	**0.90***
jCLUS	14.26	7.21	19.37	8.16	7.21	0.85
eCLUS	28.86	17.81	33.96	11.68	8.61	0.83
aCLUS	17.42	9.74	22.37	8.63	7.37	0.85
rCLUS	-	-	-	-	-	-
pCLUS	22.69	14.85	27.06	9.77	8.21	0.85
tCLUS	15.68	8.54	21.25	8.58	7.19	*0.87*
hospital	2308.00	637.00	3966.00	504.00	624.00	0.49
CoTraDiC	1042.49	321.81	1224.30	261.38	493.56	**0.49***
jCLUS	1754.99	474.32	2424.40	371.67	537.65	0.43
eCLUS	32581.16	373.22	1402.34	295.59	495.93	0.43
aCLUS	2290.07	631.12	3923.52	499.55	619.71	*0.49**
rCLUS	1106.98	388.57	2014.11	307.53	459.74	0.48
pCLUS	32581.16	373.22	1402.34	295.59	495.93	0.43
tCLUS	2292.86	626.64	3949.00	500.69	623.54	*0.49**
isbpm	138.00	58.00	234.00	87.00	110.00	0.99
CoTraDiC	69.32	28.26	129.96	50.31	61.66	**1.00***
jCLUS	90.64	38.06	161.73	61.84	75.61	0.99
eCLUS	90.64	38.06	161.73	61.84	75.61	0.99
aCLUS	90.64	38.06	161.73	61.84	75.61	0.99
rCLUS	90.64	38.06	161.73	61.84	75.61	0.99
pCLUS	73.78	30.67	146.22	57.44	69.55	*1.00**
tCLUS	66.88	27.95	120.76	46.40	56.90	0.99
photo	138.00	84.00	221.00	85.00	76.00	0.79
CoTraDiC	124.00	83.00	204.50	80.50	67.50	**0.86***
jCLUS	79.84	51.52	140.98	56.83	48.77	0.85
eCLUS	79.84	51.52	140.98	56.83	48.77	0.85
aCLUS	79.84	51.52	140.98	56.83	48.77	*0.85*
rCLUS	129.04	79.01	206.42	79.70	71.38	0.83
pCLUS	133.24	79.24	215.24	82.92	76.00	0.81
tCLUS	79.84	51.52	140.98	56.83	48.77	*0.85*
repair	13.00	10.00	30.00	12.00	12.00	0.89*
CoTraDiC	19.12	12.06	32.06	12.58	12.00	**0.88**
jCLUS	17.30	12.01	31.43	12.57	12.00	**0.88**
eCLUS	21.93	13.85	33.08	12.62	12.00	**0.88**
aCLUS	10.04	6.75	26.47	12.00	11.91	*0.98*
rCLUS	12.78	9.57	28.91	11.78	11.57	0.88
pCLUS	12.78	9.57	28.91	11.78	11.57	0.88
tCLUS	15.22	10.89	30.27	12.20	11.79	0.87
review	25.00	11.00	44.00	17.00	20.00	1.00*
CoTraDiC	22.84	11.00	41.84	16.46	18.92	**1.00***
jCLUS	25.00	11.00	44.00	17.00	20.00	**1.00***
eCLUS	27.98	13.98	46.98	18.49	20.00	0.99
aCLUS	27.98	13.98	46.98	18.49	20.00	0.99
rCLUS	35.48	16.44	49.92	18.96	20.00	0.99
pCLUS	26.12	12.12	45.12	17.56	20.00	0.99
tCLUS	25.00	11.00	44.00	17.00	20.00	*1.00**
lReview	24.00	11.00	44.00	17.00	20.00	1.00*
CoTraDiC	24.00	11.00	44.00	17.00	20.00	**1.00***
jCLUS	35.39	21.47	53.49	19.99	20.00	0.95
eCLUS	27.26	13.79	46.98	18.49	20.00	0.98
aCLUS	34.50	17.01	50.01	18.50	20.00	0.99
rCLUS	24.00	11.00	44.00	17.00	20.00	*1.00**
pCLUS	26.73	13.34	45.56	17.39	20.00	0.98
tCLUS	20.00	11.00	40.00	17.00	20.00	*1.00**
claims	221.00	20.00	63.00	17.00	21.00	0.92
CoTraDiC	87.00	13.00	43.00	14.00	17.00	**0.94**
jCLUS	87.00	13.00	43.00	14.00	17.00	**0.94**
eCLUS	87.00	13.00	43.00	14.00	17.00	**0.94**
aCLUS	87.00	13.00	43.00	14.00	17.00	0.94
rCLUS	87.00	13.00	43.00	14.00	17.00	0.94
pCLUS	45.17	9.31	40.57	15.22	17.92	*0.96**
tCLUS	87.00	13.00	43.00	14.00	17.00	0.94

The baseline process model is the Petri net computed from the entire event log. It is compared to the Petri nets computed from the trace clustering pattern, discovered by considering the multiple trace profiles (CoTraDiC, jCLUS and eCLUS), as well as from the trace clustering pattern, discovered by considering a single trace profile - activity (aCLUS), resource (rCLUS), performance (pCLUS) and transition (tCLUS). The best fitness computed with a multiple view clustering pattern is in bold. The best fitness computed with a single-view clustering pattern is in italics. (*) identifies the Petri net model with the best fitness per log.

Results on conformance show that a clustering phase almost always improves or, at least, preserves conformance of the process models which are then computed. By focusing the analysis on the multiple view algorithms, we can observe that clustering patterns computed by CoTraDiC have more (or at least equal) conformance than clustering patterns, computed by both jCLUS and eCLUS for ten out of 11 event logs. The exception is the event log oProblem, where both jCLUS and eCLUS gain more conformance than CoTraDiC. However, also in this event log, the Petri nets, computed from the traces clustered by CoTraDiC, have more conformance than the baseline Petri net, which is computed from the traces of the entire event log, as well as more conformance than the Petri nets, which are computed from the traces clustered by the single view algorithms (aCLUS, rCLUS, pCLUS and tCLUS). On the other hand, clustering patterns, computed by CoTraDiC, gain the most conformance for eight out of 11 event logs of this comparative study. The exceptions are the event logs: oProblem, repair and claims where the most conformance is gained by eCLUS, aCLUS and pCLUS, respectively. Thus, also when the multiple view learning with cotraining strategy is outperformed by single learning algorithms, we cannot see the best trace profile to be used for clustering. In fact, by focusing the analysis on the single view algorithms, we can observe that the conformance of clustering patterns extracted from the view Activity is more than (or equal to) the conformance of clustering patterns extracted from views Resource, Performance or Transition for five out 11 event logs, the conformance of clustering patterns extracted from the view Resource is more than (or equal to) conformance of clustering patterns extracted from views Activity,



(a) Cluster 1 (200 traces)

(b) Cluster 2 (275 traces)

Fig. 4. Petri nets (etm). Process models computed from both a cluster of 200 etm traces (4a) and a cluster of 275 etm traces (4b), respectively. Both clusters are computed by using CoTraDiC. The process model computed from the entire log of 475 traces is shown in Fig. 1.

Performance or Transition for two out 11 event logs, the conformance of clustering patterns extracted from the view Performance is more than (or equal to) conformance of clustering patterns extracted from views Activity, Resource or Transition for three out 11 event logs, while conformance of clustering patterns extracted from the view Transition is more than (or equal to) conformance of clustering patterns extracted from views Activity, Resource or Performance for six out 11 event logs. This shows that there is no trace profile (or equivalently no process mining perspective) that is more informative than others in absolute. This result can be interpreted as a validation of the viability of both multiple view clustering and co-training strategy in process mining.

Results on complexity show that the complexity of the Petri nets, which are computed from the clustered traces, is generally lower than the complexity of the baseline Petri nets, which are computed from the traces of the entire log. Complexity metrics almost always testify to a reduction of Petri nets complexity when traces are clustered by CoTraDiC. The only exception is the event log repair, where, on the other hand, no multiple view clustering pattern is able to reduce the complexity of the baseline Petri net. To conclude this analysis of the complexity of cluster-based Petri nets, we analyze the Petri nets computed from the event log etm. We consider the traces clustered by CoTraDiC. Petri nets, computed from the clustered traces are shown in Figs. 4a and 4b, while the baseline Petri net, computed from the traces of the event log, is shown in Fig. 1. We can observe that, in this case, clustering has appropriately clustered together two hundred related traces (Cluster 1, Fig. 4a), by allowing us to discover a standard process model that is significantly more comprehensible than the baseline model (Fig. 1). On the other hand, the remaining two hundred and seventy-five clustered traces allow us to discover a Petri net (Cluster 2, Fig. 4b) that is no more complex than the baseline Petri net (Fig. 1) and captures several exception of the standard process model of Cluster 1.

7 Conclusion

Traditional process mining algorithms have problems dealing with unstructured processes and generate spaghetti-like process models that are hard to understand. An approach to overcome this problem is to cluster process executions (traces), such that each of the resulting clusters corresponds to a coherent set of traces that can be adequately represented by a comprehensible process model. In this paper, we consider that traces of an event log can be represented in multiple trace profiles, derived by accounting for the several perspectives (activity, control flow, organization and performance) investigated in process mining [1]. We describe an algorithm, in order to learn these multiple trace profiles by co-training. This allows us to generate a single clustering pattern according to all perspectives. The empirical evaluation shows that the traces clustered with the co-training strategy have high compactness when compared to traces clustered with alternative multiple view learning approaches. In addition, they allow us to compute process models that have high conformance and comprehensibility when compared to the process model, discovered from the traces of the entire event log, as well as the process models, discovered from the traces clustered with both alternative multiple view and baseline single view approaches. This supports the idea that processing jointly the multiple perspectives of trace data allows us to minimize the problem of spaghetti-like process models, by handling variability in the recorded behavior of existing logs and facilitating process model discovery. It also shows that co-training is an efficacious strategy for multiple view learning in process mining.

Some directions for further work are still to be explored. A weighting schema can be used, in order to weigh the relative importance of the specific trace profile in the co-training strategy. Moreover, it would be interesting to investigate model-based stopping criteria for the co-training process, which account for the conformance/complexity of the process models discovered with the clustered traces. Finally, the defined strategy can be easily parallelized. We intend to investigate this opportunity, in order to reduce both time and space complexities of the computations.

Acknowledgments

This work fulfills the research objectives of the PON 02_00563_3470993 project “VINCENTE—A Virtual collective INtelligenCe ENvironment to develop sustainable Technology Entrepreneurship ecosystems”, funded by the Italian Ministry of University and Research (MIUR), as well as the ATENEO 2012 project “Mining Complex Patterns” funded by University of Bari Aldo Moro. The authors thank Wil van der Aalst, Lijie Wen, and Qinlong Guo for providing their support to use Alpha++, Jorge Munoz-Gama for explaining the measures of the conformance analysis in ProM and Lynn Rudd for her help in reading the manuscript.

References

[1] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. 1st ed., New York, NY, USA: Springer, 2011.

[2] A. Kalsing, G. do Nascimento, C. Iochpe, and L. Thom, “An incremental process mining approach to extract knowledge from legacy systems,” in *Proc. 14th IEEE Int. Conf. Enterprise Distrib. Object Comput.*, 2010, pp. 79–88.

[3] R. Bose and W. Aalst, “Trace clustering based on conserved patterns: Towards achieving better process models,” in *Proc. Bus. Process Manag. Workshops*, 2010, pp. 170–181.

[4] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca, "Discovering expressive process models by clustering log traces," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 8, pp. 1010–1027, Aug. 2006.

[5] M. Song, C. Ganther, and W. Aalst, "Trace clustering in process mining," in *Proc. Bus. Process Manag. Workshops*, 2009, vol. 17, pp. 109–120.

[6] R. P. J. C. Bose and W. M. P. van der Aalst, "Context aware trace clustering: Towards improving process mining results," in *Proc. SIAM Int. Conf. Data Mining*, 2009, pp. 401–412.

[7] D. Ferreira, M. Zacarias, M. Malheiros, and P. Ferreira, "Approaching process mining with sequence clustering: Experiments and findings," in *Proc. 5th Int. Conf. Bus. Process Manag.*, 2007, pp. 360–374.

[8] J. De Weerdt, S. vanden Broucke, J. Vanthienen, and B. Baesens, "Active trace clustering for improved process discovery," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 12, pp. 2708–2720, Dec. 2013.

[9] F. Folino, G. Greco, A. Guzzo, and L. Pontieri, "Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction," *Data Knowl. Eng.*, vol. 70, no. 12, pp. 1005–1029, 2011.

[10] J. Yu, M. Wang, and D. Tao, "Semisupervised multiview distance metric learning for cartoon synthesis," *IEEE Trans. Image Process.*, vol. 21, no. 11, pp. 4636–4648, Nov. 2012.

[11] S. Bickel and T. Scheffer, "Multi-view clustering," in *Proc. 4th IEEE Int. Conf. Data Mining*, 2004, pp. 19–26.

[12] K. Chaudhuri, S. M. Kakade, K. Livescu, and K. Sridharan,, "Multi-view clustering via canonical correlation analysis," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 129–136.

[13] X. Cai, F. Nie, and H. Huang, "Multi-view k-means clustering on big data," in *Proc. 23rd Int. Joint Conf. Artif. Intell.*, 2013, pp. 2598–2604.

[14] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proc. 11th Annu. Conf. Comput. Learn. Theory*, 1998, pp. 92–100.

[15] S. Kiritchenko and S. Matwin, "Email classification with co-training," in *Proc. Conf. Centre Adv. Stud. Collaborative Res.*, 2001, p. 8.

[16] D. Malerba, M. Ceci, and A. Appice, "A relational approach to probabilistic classification in a transductive setting," *Eng. Appl. AI*, vol. 22, no. 1, pp. 109–116, 2009.

[17] M. Ceci, A. Appice, H. Viktor, D. Malerba, E. Paquet, and H. Guo, "Transductive relational classification in the co-training paradigm," in *Proc. 8th Int. Conf. Mach. Learni. Data Mining Pattern Recog.*, 2012, pp. 11–25.

[18] A. Kumar and H. Daumé, "A co-training approach for multi-view spectral clustering," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 393–400.

[19] S. Sun, "A survey of multi-view machine learning," *Neural Comput. Appl.*, vol. 23, no. 7/8, pp. 2031–2038, 2013.

[20] B. Long, P. S. Yu, and Z. M. Zhang, "A general model for multiple view unsupervised learning," in *Proc. SIAM Int. Conf. Data Mining*, 2008, pp. 822–833.

[21] V. R. de Sa, "A spectral clustering with two views," in *Proc. ICML Workshop Learn. Multiple Views*, 2005, pp. 20–27.

[22] B. Long, X. Wu, Z. M. Zhang, and P. S. Yu, "Unsupervised learning on k-partite graphs," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2006, pp. 317–326.

[23] D. Zhou and C. J. C. Burges, "Spectral clustering and transductive learning with multiple views," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 1159–1166.

[24] G. Tzortzis and A. Likas, "Convex mixture models for multi-view clustering," in *Proc. 19th Int. Conf. Artif. Neural Netw.*, 2009, pp. 205–214.

[25] G. Cleuziou, M. Exbrayat, L. Martin, and J. Sublemontier, "CoFKM: A centralized method for multiple-view clustering," in *Proc. 9th IEEE Int. Conf. Data Mining*, 2009, pp. 752–757.

[26] H. Tao, C. Hou, and D. Yi, "Multiple-view spectral embedded clustering using a co-training approach," in *Proc. Int. Conf. Comput. Eng. Netw.*, 2014, pp. 979–987.

[27] A. Strehl and J. Ghosh, "Cluster ensembles—a knowledge reuse framework for combining multiple partitions," *J. Mach. Learn. Res.*, vol. 3, pp. 583–617, 2003.

[28] Y. Cheng and R. Zhao, "Multiview spectral clustering via ensemble," in *Proc. IEEE Int. Conf. Granular Comput.*, 2009, pp. 101–106.

[29] A. L. N. Fred and A. Jain, "Data clustering using evidence accumulation," in *Proc. 16th Int. Conf. Pattern Recog.*, 2002, vol. 4, pp. 276–280.

[30] W. Qian and A. Zhou, "Analyzing popular clustering algorithms from different viewpoints," *Int. J. Softw.*, vol. 13, pp. 1382–1394, 2002.

[31] P. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, no. 1, pp. 53–65, Nov. 1987.

[32] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *J. Symb. Comput.*, vol. 9, no. 3, pp. 251–280, 1990.

[33] B. van Dongen, B. Weber, D. R. Ferreira, and J. De Weerdt, "Summary of the BPI Challenge 2013," in *Proc. CEUR Workshop: 3rd Bus. Process Intell. Challenge Co-Located 9th Int. Bus. Process Intell. Workshop*, 2013, vol. 10–52, pp. 1–9.

[34] B. van Dongen, B. Weber, D. R. Ferreira. (2011). Summary of the BPI Challenge 2011, in *Proc. 1st Business Process Intelligence Challenge Co-Located 7th Int. Bus. Process Intell. Workshop* [Online]. Available: http://www.win.tue.nl/bpi/2011/challenge

[35] R. P. J. C. Bose, H. M. W. E. Verbeek, and W. M. P. van der Aalst, "Discovering hierarchical process models using prom," in *Proc. CAiSE Forum*, 2011, pp. 33–40.

[36] L. Wen, W. M. Aalst, J. Wang, and J. Sun, "Mining process models with non-free-choice constructs," *Data Mining Knowl. Discovery*, vol. 15, no. 2, pp. 145–180, 2007.

[37] J. Mendling and M. Strembeck, "Influence factors of understanding business process models," in *Proc. 11th Int. Conf. Bus. Inf. Syst.*, 2008, pp. 142–153.

[38] L. Kaufman and P. Rousseeuw, "Clustering by means of medoids," in *Statistical Data Analysis Based on the L1-Norm and Related Methods*. Amsterdam, The Netherland: North-Holland, 1987, pp. 405–416.

[39] M. Borgo, A. Soranzo, and M. Grassi, *MATLAB for Psychologists*. New York, NY, USA: Springer, 2012.

Annalisa Appice is an assistant professor in the Department of Informatics, University of Bari Aldo Moro. Her research activity mainly concerns data mining and machine learning. She has published more than 120 papers in international journals and conference proceedings. She is involved in many European and national projects on data mining. She has served in the program committee of more than 30 international conferences and workshops.

Donato Malerba is a full professor in the Department of Computer Science, University of Bari Aldo Moro. His research activity mainly concerns data mining, machine learning, and big data. He has published more than 200 papers in international journals and conference proceedings. He received the IBM Faculty Award for the year 2004. He is responsible for a research unit of several European and national projects. He is a member of the IEEE.