# Context Aware Trace Clustering: Towards Improving Process Mining Results

R. P. Jagadeesh Chandra Bose*† Wil M.P. van der Aalst*

**Abstract**

Process Mining refers to the extraction of process models from event logs. Real-life processes tend to be less structured and more flexible. Traditional process mining algorithms have problems dealing with such unstructured processes and generate spaghetti-like process models that are hard to comprehend. An approach to overcome this is to cluster process instances (a process instance is manifested as a trace and an event log corresponds to a multi-set of traces) such that each of the resulting clusters correspond to a coherent set of process instances that can be adequately represented by a process model. In this paper, we propose a context aware approach to trace clustering based on generic edit distance. It is well known that the generic edit distance framework is highly sensitive to the costs of edit operations. We define an automated approach to derive the costs of edit operations. The method proposed in this paper outperforms contemporary approaches to trace clustering in process mining. We evaluate the goodness of the formed clusters using established fitness and comprehensibility metrics defined in the context of process mining. The proposed approach is able to generate clusters such that the process models mined from the clustered traces show a high degree of fitness and comprehensibility when compared to contemporary approaches.

## 1 Introduction

Process mining techniques can deliver valuable, factual insights into how processes are being executed in real life. Process mining refers to the extraction of process models from event logs [1]. An event log corresponds to a bag of process instances of a business process. A process instance is manifested as a trace (a trace is defined as an ordered list of activities invoked by a process instance from the beginning of its execution to the end). Real-life processes tend to be less structured and more flexible. Traditional process mining algorithms have problems dealing with such unstructured processes and generate spaghetti-like process models that are hard to comprehend. This is caused by the application of discovery algorithms without preprocessing raw traces. Since traces are captured for each execution of the system, there can be instances where the system is subjected to similar execution patterns/behavior, and instances where unrelated cases are executed. Considering the set of traces in the event log all at once might lead to ambiguities for the mining algorithms which often result in spaghetti-like models. An approach to overcome this is to cluster the traces such that each of the resulting clusters corresponds to a coherent set of cases that can be adequately represented by a process model. Figure 1 illustrates the significance of trace clustering in process mining. The process model on the top right of Figure 1 is a process model mined from the entire event log. The model is quite complex to comprehend. The bottom rectangle of Figure 1 depicts the process models mined from clustered traces. It is evident that clustering enables the comprehension of process models by reducing the spaghetti-ness.

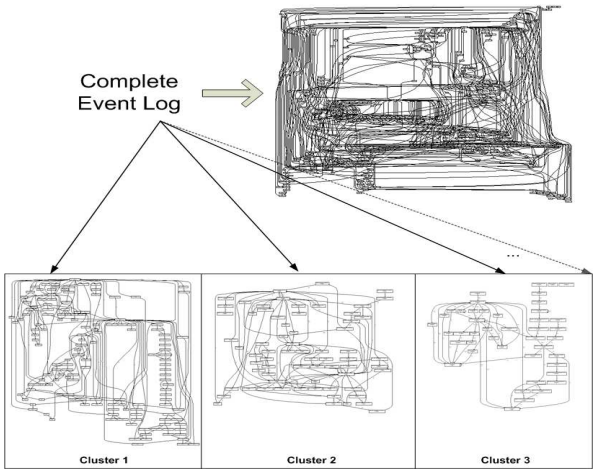



Figure 1: Significance of trace clustering in Process Mining

The basic principle in clustering is to define a notion of similarity or dissimilarity between traces and then partition the event log into $k$-clusters (for some $k \geq 2$)

*Department of Mathematics and Computer Science, University of Technology, Eindhoven, The Netherlands

†Philips Healthcare, Veenpluis 4-6, 5684 PC Best, The Netherlands

such that all traces within a cluster $C_i$ are similar in some sense, and traces belonging to two different clusters are dissimilar. The goodness of the formed clusters is largely dependent on the notion of similarity used. A poor choice of similarity metric can lead to bad clusters. We define two notions of goodness from a process mining perspective. Process mining on traces from good clusters should generate models that have a (i) high degree of fitness and (ii) low degree of structural complexity (less spaghetti like). Fitness quantifies how much of the observed behavior is captured in the model.

Traditional approaches (to trace clustering) in the literature were confined to transforming the traces into a vector space and using a pool of clustering techniques (agglomerative hierarchical clustering, k-means clustering etc) with different distance metrics in the vector space (Euclidean distance, Jaccard distance etc). In this paper, we propose a new approach to trace clustering based on generic edit-distance and show that the proposed approach outperforms other approaches in that it partitions the traces into clusters such that the process models mined from those clusters show a high degree of fitness and that the models are more comprehensible. Edit-distance is sensitive to the cost function (of edit operations). In this paper, we define a method to automatically derive the cost function and show that the cost function thus derived has semantic significance.

Philips Healthcare looks at process mining as a tool to deliver a powerful set of solutions for providing factual and appropriate insights into their product usage, and believes that the insights gained would enable them to build efficient and customer-focused product designs and maintenance processes. Philips Healthcare collates logs from their medical systems across the globe. These logs contain information about user actions, system events etc. The number of such log-recording systems in conjunction with the fine grained nature of logging makes the data available not just huge, but massively huge. Trace clustering assumes utmost importance in dealing with such voluminous data.

The rest of the paper is organized as follows. Section 2 defines the notations used in the paper. In Section 3, we discuss various approaches to trace clustering by highlighting the advantages and pitfalls of each. In Section 4, we present an algorithm to derive the cost function for the generic edit distance framework. Section 5 presents the clustering approach and introduces the metrics used to evaluate the goodness of clusters. In Section 6, we present and discuss the experimental results. Related work is presented in Section 7. Finally, Section 8 concludes with remarks on future directions.

## 2 Notations

Let $\mathcal{A}$ denote the set of activities. $|\mathcal{A}|$ is the number of activities.
$\mathcal{A}^+$ is the set of all non-empty finite sequences of activities from $\mathcal{A}$. A trace, $T$ is an element of $\mathcal{A}^+$.
The set of all $n$-length sequences over the alphabet $\mathcal{A}$ is denoted by $\mathcal{A}^n$. A trace of length $n$ is denoted as $T^n$ i.e., $T^n \in \mathcal{A}^n$, and $|T^n| = n$.
The ordered sequence of activities in $T^n$ is denoted as $T(1)T(2)T(3)\ldots T(n)$ where $T(k)$ represents the $k^{th}$ activity in the trace. Alternatively, for readability purposes, we also denote the trace $T^n$ as an ordered list of activities $(T(1), T(2), T(3), \ldots, T(n))$.
$T^{n-1}$ denotes the subsequence of $T^n$ with the first $n-1$ activities. In other words $T^n = T^{n-1}T(n)$.
A trace, $T$, without a superscript denotes an arbitrary length trace, i.e., $T \in \mathcal{A}^+$.
An event log, $\mathcal{L}$, corresponds to a multi-set (or bag) of traces from $\mathcal{A}^+$.

As an example, let $\mathcal{A} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ be the set of activities; $|\mathcal{A}| = 3$. $T$ =`abcabb` is a trace of length 6. $T(1)$=`a`, $T(2)$=`b`, $T(3)$=`c`, $T(4)$=`a`, $T(5)$=`b`, $T(6)$=`b`. $T(2,5)$ =`bcab` is a subsequence of $T$ from positions 2 to 5. $\mathcal{L} = \{\mathtt{aba}, \mathtt{aba}, \mathtt{abba}, \mathtt{baca}, \mathtt{acc}, \mathtt{cac}\}$ represents an event log.

## 3 Approaches to Trace Clustering: Issues and Challenges

**3.1 Bag-of-activities approach** One of the most often used techniques for analyzing (clustering) traces is to transform a trace into a vector, where each dimension of the vector correspond to an activity [6], [7], [12]. The set of all activities present in the event log defines the number of dimensions of the vector. For each trace, the values of the vector correspond to the frequency count of the activities in that trace. For example, the traces `abaac` and `badca` correspond to the vectors [3, 1, 1, 0] and [2, 1, 1, 1] respectively; the dimensions of the vector being [`a,b,c,d`]. Similarity between traces is then estimated using the standard distance metrics (such as the Euclidean distance) in the vector-space model. This transformation, referred to as a *bag-of-activities* transformation has a few drawbacks:

1. *Lack of context information:* Process execution is characterized by a context. The bag-of-activities representation does not capture the dynamics of process execution. As an example, consider a process model with a notification activity. The

different instances of notification within a trace might have different connotations based on the context in which it is invoked. For example, a broad-cast notification, notification of information, notification requesting a response.

2. *Order of execution:* The bag-of-activities representation also loses the information on the order of execution of events. Any permutation of the bag of activities of a given trace has the same vector representation and thereby has a distance of 0 with each other. However, in reality, a lot of these permutations do not make any sense from a process definition point of view. For example, one cannot `write` to a file until the file is `opened`. Even in cases where it makes sense, they might represent two different use-cases.

**3.2 k-gram model** One means of incorporating context into the vector space model is to consider subsequences of activities. These subsequences capture the order of execution as well. However, it is important to note that the notion of context can be much more than a mere order of execution of activities. Henceforth, we refer to a subsequence of $k$ activities as $k$-gram. For the trace `abacaab`, the set of $2-$grams correspond to {`ab,ba,ac,ca,aa`} while the set of 3-grams correspond to {`aba,bac,aca,caa,aab`}. One can transform a trace into a vector in the $k$-gram model, which now incorporates certain context information. The clustering of traces is then performed in the $k-$gram space. However, it is to be noted that the size of this model increases drastically as the size of the alphabet $|\mathcal{A}|$ and $k$ increase. For a system with 100 activities and considering 3-grams, we end up with potentially $100^3 = 10^6$ dimensions. In reality, one may not see all combinations of 3-grams in the event log; thus the number of dimensions would be less than $10^6$. Nonetheless, working in the $k$-gram space incurs a huge computational overhead. In addition, selecting a suitable value for $k$ is non-trivial.

**3.3 Hamming Distance** While the vector-space model falls under the statistical processing domain, Hamming distance and edit-distance [2] are two of the most often used syntactic methods in text mining to quantify the (dis-)similarity of two words/sequences. Hamming distance, defined for two sequences of equal length measures the count of character positions in which the two sequences differ. For the sequences `abacaab` and `abcaaba`, the Hamming distance is equal to 4, since the two sequences differ at positions $3, 4, 6$ and 7. One can adopt Hamming distance to event log traces; instead of counting the character positions that differ, one now needs to count the activities that differ at a position. This notion though useful in certain cases, is not flexible enough for a majority of event traces due to the following reasons:

i. Hamming distance is not defined for sequences of different lengths. In reality, event traces will have different lengths.

ii. Even in cases where Hamming distance is defined, two traces from the same process model can manifest differently. Interleaving of activities in two traces are punished too strongly in Hamming distance.

**3.4 Edit distance** Another fundamental measure of (dis-)similarity between two sequences is the Levenshtein distance, also called as edit distance. Levenshtein distance between two sequences is defined as the minimum number of edit operations needed to transform one sequence into the other, where an edit operation is an insertion, deletion or substitution of an element. Consider two sequences $S$ and $T \in \mathcal{A}^+$. $S$ and $T$ may contain (a) symbols common to both of them, (b) symbols present only in $S$ and (c) symbols present only in $T$. For example, consider the two sequences $S =$ `teach` and $T =$ `tricky`, $|S| = 5$ and $|T| = 6$. $S$ and $T$ have the symbols `t` and `c` in common. `a`, `e` and `h` are symbols present only in $S$ while `i`, `r`, `k` and `y` occur only in $T$. A transformation of sequence $S$ to sequence $T$ will be the set of editing operations applied to one of the sequences iteratively, which transform $S$ into $T$. There are many possibilities in which one can transform $S$ into $T$. One can delete symbols that occur only in $S$ and insert symbols that occur only in $T$ or one can replace certain symbols in $S$ with symbols in $T$.

For two sequences $S$ and $T$, the following edit operations are considered on the alphabet $\mathcal{A} \cup \{-\}$, where $-$ denotes a gap. For $a, b \in \mathcal{A}$, the pair

- $(a, a)$ denotes a match of symbols between $S$ and $T$ at some position $S(i)$ and $T(j)$. A match can be considered as a substitution of a symbol with itself.
- $(a, -)$ denotes the deletion of $a$ in $S$ at some position $S(i)$
- $(-, b)$ denotes the insertion of $b$ in $S$
- $(a, b)$ denotes the replacement/substitution of $a$ in $S$ with $b$ at some position $S(i)$ where $a \neq b$

For the above example sequences $S$ and $T$, the following sequence of edit operations can transform $S$ into $T$:

(t,t)(-,r)(e,-)(a,i)(c,c)(h,k),(-,y). The edit distance framework works by assigning a cost or weight for each of the edit operations defined above. The advantage of using edit distance is that it considers a trace in totality thereby preserving the context and ordering.

More formally, the *generic string edit distance* can be characterized by a triple $< \mathcal{A}, \mathcal{B}, c >$[1] consisting of finite alphabets $\mathcal{A}$ and $\mathcal{B}$ and the primitive cost function $c : E \rightarrow \Re^+$ where $E = E_d \cup E_i \cup E_d$ is the set of primitive edit operations on the alphabets and $\Re^+$ is the set of nonnegative real numbers. $E_s = \mathcal{A} \times \mathcal{B}$ is the set of substitutions, $E_d = \mathcal{A} \times \{-\}$ is the set of deletions, and $E_i = \{-\} \times \mathcal{B}$ is the set of insertions. The distance between two strings $S^m$ and $T^n$, $S^m \in \mathcal{A}^m$, $T^n \in \mathcal{B}^n$ $(m \geq 1, n \geq 1)$, can be defined as:

(3.1)

$$d_c(S^m, T^n) = min \begin{cases} c(S(m), T(n)) + d_c(S^{m-1}, T^{n-1}), \\ c(S(m), -) + d_c(S^{m-1}, T^n), \\ c(-, T(n)) + d_c(S^m, T^{n-1}) \end{cases}$$

When either $m = 0$ or $n = 0$, only insertion/deletion operations are defined. We denote $S^0 = T^0 = -$; Thus

$$\begin{aligned} d_c(S^m, -) &= c(S(m), -) + d_c(S^{m-1}, -), m \geq 1 \\ d_c(-, T^n) &= c(-, T(n)) + d_c(-, T^{n-1}), n \geq 1 \\ d_c(-, -) &= 0; \end{aligned} \tag{3.2}$$

*The Levenshtein distance is a specific case of the generic edit distance.* In the Levenshtein distance, a unit cost model is used for the edit operations. In other words, under Levenshtein distance, $c(\mathtt{a}, \mathtt{a}) = 0$, $c(\mathtt{a}, -) = c(-, \mathtt{a}) = 1$, and $c(\mathtt{a}, \mathtt{b}) = 1$ for $\mathtt{a} \neq \mathtt{b}$. Consider two strings $S = \mathtt{abcac}$ and $T = \mathtt{acacad}$. The Levenshtein distance between $S$ and $T$ is 3. The sequence of edit operations transforming $S$ to $T$ can be visualized as an alignment between $S$ and $T$ and is depicted in Figure 2.

S: a b c a c - -
T: a - c a c a d

Figure 2: Sequence of edit operations transforming S to T depicted as an alignment

Levenshtein distance, though noteworthy for its simplicity, does not fit in many application scenarios. Consider our case of event log traces, and the following scenarios in diagnosing a patient using a CardioVascular medical system:

- For the event traces $T_1$ = (`SetPhysician`, `SetPatientType`, `StartExamination`, `AddExamination`, `StopExamination`) and $T_2$ = (`SetPatientType`, `SetPhysician`, `SetOperatorName`, `StopExamination`, `AddExamination`), the Levenshtein distance would be 5. Figure 3 depicts two transformations with Levenshtein distance of 5. However it is to be noted, from an application point of view, the sequence of events in traces $T_1$ and $T_2$ can be abstracted to `Presetting` and `Examination` functionality. The `Set` commands belong to `Presetting` while the rest to `Examination`. Also (a) it really may not matter whether the `Physician` is set first or the `Patient` (b) one cannot `Stop` an examination that is not `Started` and (c) all examination commands should be enclosed between `Start` and `Stop` commands.

- Now, consider another trace $T_3$ = (`MoveDetectorFrontal`, `AngulateBeamFrontal`, `RotateBeamFrontal`, `SelectInjectorControl`). The Levenshtein distance between $T_1$ and $T_3$ is 5. The Levenshtein distance between $T_2$ and $T_3$ is also 5. Although $T_3$ represents a characteristically distinct functionality when compared to $T_1$ and $T_2$, clustering based on the Levenshtein distance is likely to put all the three traces in a single cluster.

In other words, Levenshtein distance does not consider the functional validity of any edit operation. Also, two sequences of lengths $n_1$ and $n_2$, irrespective of their similarity, will always have a Levenshtein distance of at least $|n_1 - n_2|$, where $|n|$ denotes the absolute value of $n$. It is quite natural that event log traces would be of different length. The Levenshtein distance applied as is, would give a non-zero distance for two traces that are functionally similar. For example, consider the two traces `abacd` and `abacacacd`. These two traces are similar in that they would have been generated from the same process model where there is a loop construct over the activities `ac`. Ideally, one would like to put these two traces in the same cluster. However, if we apply Levenshtein metric, we get a distance of 4. One should consider the manifestations of process model constructs to alleviate such problems.

In order to avoid edit operations that do not make sense in a certain context, the cost function, $c$, needs to be more robust. Substitution of uncorrelated/constrasting activities or insertion/deletion of activities not confirming to a context should be penalized heavily. On the other hand, 'like' events should be allowed to be replaced/inserted at a minimal cost. However, deriving such costs is nontrivial unless provided by a domain expert. In the next section, we propose an approach

[1] In most applications, $\mathcal{A} = \mathcal{B}$

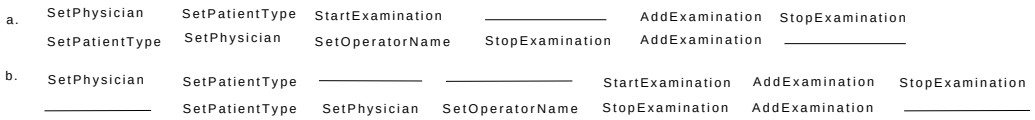



Figure 3: Different transformations with a Levenshtein distance of 5

to automatically derive the edit operation costs from event log traces and show that the derived costs have statistical as well as semantic significance.

## 4 Deriving Substitution and Indel Costs

Distance and similarity measures are interchangeable in the sense that a small distance means high similarity, and vice versa. For two sequences $S^m$ and $T^n$ ($m \geq 1, n \geq 1$), the edit distance (3.1) defined in Section 3.4 can be transformed to a similarity function defined as in (4.3).

(4.3)

$$Sim(S^m, T^n) = max \begin{cases} s(S(m), T(n)) + Sim(S^{m-1}, T^{n-1}), \\ i(S(m), -) + Sim(S^{m-1}, T^n), \\ i(-, T(n)) + Sim(S^m, T^{n-1}) \end{cases}$$

$s : (\mathcal{A} \times \mathcal{B}) \rightarrow \Re$ defines the substitution scores. $s(S(m), T(n))$ defines the score for substituting $S(m)$ with $T(n)$. $i$ defines the indel (insertion/deletion) scores. $i(S(m), -)$ defines the score for deleting $S(m)$ while $i(-, T(n))$ defines the score for inserting $T(n)$. On similar lines of (4.3), one can transform (3.2) for the base condition when either $m = 0$ or $n = 0$.

In this section, we derive the scores for substitution and insertion/deletion of symbols for *similarity* rather than the costs for *distance*. Before we discuss the algorithm, let us first lay down the characteristics that a substitution and indel scoring matrix should hold:

1. substitution of uncorrelated activities should be discouraged
2. substitution of contrasting activities should be penalized
3. insertion of activities out of context should be discouraged
4. substitution of correlated/similar activities should be encouraged in proportion to the degree of similarity

The basic idea of substitution matrix derivation is to compare the actual observed frequency of a pair of activities sharing a particular context to their expected frequency of co-occurrence if they occur independently. We use the set of 3-grams in the event log as a notion of context. In other words, we need to estimate:

1. the probability of observing an activity in the set of all contexts
2. the probability of observing a pair of activities that can occur within the same context

**4.1 Substitution Scores** Algorithm 1 presented in this section generates the substitution scores. It is to be noted that this algorithm tries to maximize the score of two sequences based on the similarity. In other words, it derives scores for substitution such that sequences that are similar attains a high score and sequences that are not similar gets a low score. The edit-distance on the other hand assigns a low value for similar sequences. We will later define a transformation between the similarity score and the distance value.

Let us discuss the fundamental steps of the algorithm (steps 2 to 5) with an example. Consider the event log, $\mathcal{L}$ = { aabcdbbcda, dabcdabcbb, bbbcdbbbccaa, aaadabbccc, aaacdcdcbedbccbadbdebdc} over the alphabet $\mathcal{A} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}\}$. The set of all 3-grams over $\mathcal{L}$ is $G_3$ = {aaa, aab, abb, aac, aad, abc, bad, bbb, bbc, bcd, bed, caa, cba, cbb, cda, cdb, dab, dbb, dbc, dbd, ebd,...}. The corresponding frequencies of the 3-grams is represented by the vector $F_3 = [2, 1, 1, 1, 1, 3, 1, 2, 4, 4, 1, 1, 1, 1, 2, 2, 3, 2, 1, 1, 1, \ldots]$. The set of contexts of symbol a, $\mathcal{X}_{\mathtt{a}}$ = {aa, ab, ac, ad, bd, ca, db}. Similarly, the set of contexts for symbol b, $\mathcal{X}_{\mathtt{b}}$ = {ab, ac, bb, bc, ca, cb, db, dc, dd, ed}. $\mathcal{X}_{(\mathtt{a},\mathtt{b})}$ = {ab, ac, ca, db}. $\mathcal{X}_{(\mathtt{a},\mathtt{b})}$ signifies the set of all contexts common to a and b. To calculate the count of co-occurrence combinations (step 5 of Algorithm 1), we need to consider the frequency of 3-grams in the entire event log. To calculate, $C_{\mathtt{db}}(\mathtt{a}, \mathtt{b})$, we need to consider the 3-grams with db as the context for symbols a and b. In other words, we need to consider the 3-grams dab and dbb. Now, we have 3 occurrences of dab and 2 occurrences of dbb in the event log, $\mathcal{L}$. Each occurrence of the activity a can have a co-occurrence with each occurrence of b in the context db as shown in Figure 4(a). The count of co-occurrence combinations for this case is 6. Similarly, to calculate $C_{\mathtt{db}}(\mathtt{a}, \mathtt{a})$, we need to consider the 3-gram dab. There are 3 occurrences of dab in the event log. Each occurrence of the activity

| Command 1 | Command 2 | Substitution Score |
|---|---|---|
| BLObjectShuttersStop | StopStepImgFwd | -25 |
| BLWedge2RotateClockwise | BLWedge2Reset | 24 |
| BLWedge2RotateClockwise | BLCloseShutters | -4 |
| StartStepImgFwd | StopStepImgFwd | -30 |
| AddAnnotation | ShowFullScreen | 19 |
| CatheterEditBox | SwitchToAnalysis | 46 |

Table 1: Substitution scores for commands

| Command 1 | Command 2 | Substitution Score |
|---|---|---|
| Repair(Simple)-start | Repair(Complex)-start | 5 |
| Repair(Simple)-start | Repair(Simple)-start | 9 |
| TestRepair-complete | ArchiveRepair-complete | -11 |
| InformUser-complete | ArchiveRepair-complete | 0 |

Table 2: Substitution scores for a few activity pairs of the telephone repair process

`a` in the context `db` can co-occur with every other occurrence of `a` other than itself as shown in Figure 4(b). Thus, the count of co-occurrence combinations for this case is 3.

In other words, for the two symbols under consideration in a given context, each occurrence of the 3-gram of one symbol in the given context can co-occur with each occurrence of the 3-gram of the other symbol in the same context. In general, if the estimation of co-occurrence combinations is for 'like' symbols, then the count of such combinations $C_{\mathtt{xy}}(\mathtt{a},\mathtt{a}) = \binom{n}{2} = \frac{n(n-1)}{2}$, where $n$ is the frequency of the 3-gram `xay`. The count of co-occurrence combinations for 'unlike' symbols $C_{\mathtt{xy}}(\mathtt{a},\mathtt{b}) = n_i.n_j$ where $n_i$ and $n_j$ correspond to the frequency of the 3-grams `xay` and `xby` respectively.

Proceeding further, one can estimate the count of co-occurrence combinations of two symbols over all contexts thereby completing step 6 of the algorithm. Steps 7 and 8 of the algorithm normalize the counts thus calculated for every pair of symbols. Step 9 calculates the probability of occurrence of every symbol in the alphabet while Step 10 calculates the normalized co-occurrence frequencies by chance (random). Step 11 computes the ratio of the actual frequency divided by the chance frequency with which the pair occurs. Such a ratio compares the probability of an event occurring under two alternative hypotheses and is called a likelihood or odds ratio. Scores that are the logarithm of odds ratios are called log-odds score.
We have applied the algorithm over a large set of event traces (of real systems) over varying alphabet sizes and analyzed the resulting substitution matrices. In all the cases the algorithm mentioned above yielded

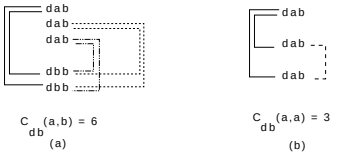



Figure 4: Count of co-occurrence combinations

substitution scores having a high semantic significance. We present one such study here where we have considered a set of 1372 event traces of a health care system. The traces correspond to the commands of clinical usage logged by the system. There were a total of 213 commands in the event traces (alphabet size, $|\mathcal{A}| = 213$). Table 1 lists the substitution scores for a few command pairs. It can be seen here that the two wedge related commands `BLWedge2RotateClockWise` and `BLWedge2Reset` assume a high score, signifying their functional closeness. The command pair, `BLWedge2RotateClockWise` and `BLCloseShutters` belonging to two different components of the system viz., `Wedge` and `Shutters` assumes a negative score. It is also important to notice that the scores are assigned relatively in proportion to their degree of closeness. Consider the scores for the command pairs, (`BLWedge2RototeClockWise`, `BLCloseShutters`) and (`BLObjectShuttersStop`, `StopStepImgFwd`). The former command pair is assigned a relatively higher score than the latter. As is obvious from the names, the former pair though belonging to different components (Wedge and Shutters), still are used for the same higher level functionality (viz., adjusting the beam). However, the latter pair where one command belongs to the shutter component and the other to an image processing operation, being totally uncorrelated

**Algorithm 1** Algorithm to derive substitution scores

1: Let $\mathcal{A}$ be the alphabet; $\mathrm{x}, \mathrm{y}, \mathrm{a}, \mathrm{b} \in \mathcal{A}$

2: Let $G_3$ denote the set of all 3-grams present in the event log and let $F_3$ denote their corresponding frequencies.

3: Define $\mathcal{X}_{\mathrm{a}}$ to be the set of all contexts of symbol a. A context of a symbol a is the subsequence xy such that $\mathrm{xay} \in G_3$

4: Define $\mathcal{X}_{(\mathrm{a},\mathrm{b})}$ to be the set of contexts common to symbols, a and b. i.e., $\mathcal{X}_{(\mathrm{a},\mathrm{b})} = \mathcal{X}_{\mathrm{a}} \cap \mathcal{X}_{\mathrm{b}}$

5: Define $C_{\mathrm{xy}}(\mathrm{a}, \mathrm{b})$ to be the count of co-occurrence combinations (explained in the description) of symbols, a and b in the given 3-gram context, $\mathrm{xy} \in \mathcal{X}_{(\mathrm{a},\mathrm{b})}$.

6: Define $C(\mathrm{a}, \mathrm{b})$ to be the count of co-occurrence combinations of symbols a and b over all contexts $\mathcal{X}_{(\mathrm{a},\mathrm{b})}$

$$C(\mathrm{a}, \mathrm{b}) = \sum_{\mathrm{xy} \in \mathcal{X}_{(\mathrm{a},\mathrm{b})}} C_{\mathrm{xy}}(\mathrm{a}, \mathrm{b})$$

7: Define $\mathcal{N}_C$ to be the norm of the count of co-occurrence combinations

$$\mathcal{N}_C = \sum_{\mathrm{a},\mathrm{b} \in \mathcal{A}} C(\mathrm{a}, \mathrm{b})$$

8: Define matrix $\mathcal{M}$ over $\mathcal{A} \times \mathcal{A}$ to be

$$\mathcal{M}(\mathrm{a}, \mathrm{b}) = [C(\mathrm{a}, \mathrm{b})/\mathcal{N}_C]$$

9: Define $p_{\mathrm{a}}$ to be the probability of occurrence of symbol $\mathrm{a} \in \mathcal{A}$

$$p_{\mathrm{a}} = \mathcal{M}(\mathrm{a}, \mathrm{a}) + \sum_{\mathrm{b} \neq \mathrm{a}} \mathcal{M}(\mathrm{a}, \mathrm{b}); \quad \sum_{\mathrm{a} \in \mathcal{A}} p_{\mathrm{a}} = 1$$

10: Define matrix $\mathbb{E}$ to be the expected value of occurrence of pair of symbols

$$\begin{aligned} \mathbb{E}(\mathrm{a}, \mathrm{b}) &= [p_{\mathrm{a}}^2] \text{ if } \mathrm{a} = \mathrm{b} \\ &= [2 p_{\mathrm{a}} p_{\mathrm{b}}] \text{ otherwise} \end{aligned}$$

11: Define the matrix of substitution scores $\mathbb{S}$ over $\mathcal{A} \times \mathcal{A}$ to be the log-odds ratio

$$\mathbb{S}(\mathrm{a}, \mathrm{b}) = \log_2 \left( \frac{\mathcal{M}(\mathrm{a}, \mathrm{b})}{\mathbb{E}(\mathrm{a}, \mathrm{b})} \right)$$

assumes a high negative score. Another important point to consider is the score for the command pair (`StartStepImgFwd`, `StopStepImgFwd`). These two commands signify contrasting operations viz., start and stop of an activity. These two contrasting commands are assigned a high negative score of $-30$ discouraging their substitution. Remember that this is one of the objectives that we started with.

As another example, let us consider the telephone repair process depicted in ProM tutorial[2]. The repair process starts by registering a telephone device sent by a customer. After registration, the telephone is sent to the Problem Detection department where it is analyzed and its defect is categorized. There are 10 different categories of defects that the phones fixed by this company can have. Once the problem is identified, the telephone is sent to the Repair department and a letter is sent to the customer to inform him/her about the problem. The Repair department has two teams. One of the teams can fix simple defects and the other team can repair complex defects. However, some of the defect categories can be repaired by both teams. Once a repair employee finishes working on a phone, this device is sent to the Quality Assurance department. There it is analyzed by an employee to check if the defect was indeed fixed or not. If the defect is not repaired, the telephone is again sent to the Repair department. If the telephone is indeed repaired, the case is archived and the telephone is sent to the customer. To save on throughput time, the company only tries to fix a defect a limited number of times. If the defect is not fixed, the case is archived anyway and a brand new device is sent to the customer. In the event log, there were a total of 12 activities for this data set. Table 2 depicts the substitution scores for a few activity pairs of this log. It is to be noted that a high positive value is assigned for activities of similar functionality. The value is also relative to the degree of similarity. For example the score for the activity pair (`Repair(Simple)Start`, `Repair(Simple)Start`) is relatively higher than for the pair (`Repair(Simple)Start`, `Repair(Complex)Start`). On the other hand, unrelated activities have a low/negative score.

One can consider not just 3-grams, but contexts of larger length as well. The basic idea of substitution matrix derivation still holds.

**4.2 Indel Scores** Event traces from a process model can have different manifestations based on the use case. The differences can be attributed to an execution of a different path or functionality or optional activities within a functionality. As an example,

[2] ProM is an extensible framework that provides a comprehensive set of tools/plugins for the discovery and analysis of process models from event logs. See http://www.processmining.org for more information and to download ProM and the dataset.

consider the sub-process 'image processing', in the process model of medical image acquisition. A lot of activities pertaining to image processing (such as zooming, filtering, segmenting) would be provided by the image processing component of the medical system. Depending on the type of patient and the diagnosis prescribed, a subset of these functionalities would be triggered. When we analyze event traces from the medical system, we see traces with variation in the usage of the image processing component. The invocation or non-invocation of certain activities can be thought of as insertion or deletion of activities in the traces. For example, consider the two traces $T_1$ = `acbcabaa` and $T_2$ = `acbcabdaa`. The difference between the two traces is that in the second trace, $T_2$, there is an invocation of activity `d` between `b` and `a`. One can transform trace $T_1$, to trace $T_2$, by inserting `d` between `b` and `a`. Alternatively, we can transform trace $T_2$ to $T_1$ by deleting activity `d`. It is important to note that insertions and deletions are complementary. Therefore, we will consider only insertions henceforth.

Insertion of activities cannot take place at random. It is natural to see insertion of activities pertaining to a functionality between activities related to the same or similar functionality than otherwise. Even within a functionality the presence/absence of an activity largely depends on its neighbors. For example, it is highly unlikely to see a image processing activity between activities pertaining to beam positioning. Similarly, it is relatively highly likely to see an edge detection activity between activities pertaining to `imageSegmentation` than between those pertaining to `imageAnnotation`. Therefore, one should have different scores for insertion of activities based on the context.

We define two kinds of insertion operations: (i) insertion of an activity to the right of an activity (ii) insertion of an activity to the left of an activity. For example, in `abc`, activity `b` can be considered as an insertion to the right of activity `a` or to the left of activity `c`. We now define an approach to determine the scores of insertion. We define two sets of scores

a. insertionRightGivenLeft

b. insertionLeftGivenRight

insertionRightGivenLeft(`a/b`) signifies the insertion of activity `a` to the right of activity `b` (or insertion of activity `a` given that the left activity is `b`). Similarly insertionLeftGivenRight(`a/c`) signifies the insertion of activity `a` to the left of activity `c` (or given that the right activity is `c`).

**Algorithm 2** Algorithm to derive insertion scores

1: Let $\mathcal{A}$ be the alphabet; $\mathrm{x}, \mathrm{y}, \mathrm{a}, \mathrm{b} \in \mathcal{A}$

2: Let $G_3$ denote the set of all 3-grams present in event log and let $F_3$ denote their corresponding frequencies.

3: Define $\mathcal{X}_\mathrm{a}$ to be the set of all contexts of symbol a. A context of a symbol a is the subsequence xy such that $\mathrm{xay} \in G_3$

4: Let $C_{\mathrm{xy}}(\mathrm{a})$ be the count of occurrences of the 3-gram $\mathrm{xay} \in G_3$

5: For each symbol $\mathrm{a}, \mathrm{x} \in \mathcal{A}$, define

$$countRightGivenLeft(\mathrm{a/x}) = \sum_{\mathrm{y}|\mathrm{xy} \in \mathcal{X}_\mathrm{a}} C_{\mathrm{xy}}(\mathrm{a})$$

6: Define

$$norm(\mathrm{a}) = \sum_{\mathrm{x} \in \mathcal{A}} countRightGivenLeft(\mathrm{a/x})$$

7: For all $\mathrm{a} \in \mathcal{A}$, let $p_\mathrm{a}$ denote the probability of occurrence of a.

8: Define

$$normCountRightGivenLeft(\mathrm{a/b}) = countRightGivenLeft(\mathrm{a/b})/norm(\mathrm{a})$$

9: The insertion scores are defined as the log-odds ratio

$$insRightGivenLeft(\mathrm{a/b}) = \log_2\left(\frac{normCountRightGivenLeft(\mathrm{a/b})}{p_\mathrm{a} p_\mathrm{b}}\right)$$

Algorithm 2 generates the scores for the insertion of activities to the right of a given activity i.e., insertionRightGivenLeft. The insertion scores for insertionLeftGivenRight can be derived in a similar fashion.

Table 3 lists a few insertion scores for the insertion of command 2 to the right of command 1. It is to be noted that related command pairs such as (`SetPatientWeight`, `SetPatientType`), (`SetContrast`, `SetEdgeGain`) assume a high positive score. The latter pair belonging to image processing functionality. It is important to closely look at the score for the command pair (`StartStepImgRev`, `StartStepImgFwd`) assuming a negative value. This signifies that it is discouraged to start an ImageForward operation immediately after starting an ImageReverse operation. The ImageReverse operation should be stopped first. This is reflected in the score for the command pair (`StopStepImgRev`,

`StartStepImgFwd`).

| Command 1 | Command 2 | insRightGivenLeftScore |
|---|---|---|
| SetPatientWeight | SetPatientType | 7 |
| BLOpenShutters | BLCloseShutters | 3 |
| StartStepRunFwd | StopStepRunFwd | 2 |
| StartStepImgRev | StartStepImgFwd | -1 |
| StopStepImgRev | StartStepImgFwd | 1 |
| SetContrast | SetEdgeGain | 4 |

Table 3: insRightGivenLeft scores for commands

The algorithms defined above derives scores for substitution/indel operations such that similar traces have a high score. One can compute the similarity between traces using these scores and take the reciprocal of that as a measure of distance. In other words for two traces $S$ and $T$, the generic edit distance, $d$, between them can be defined as

$$d(S,T) = \frac{|S| + |T|}{Sim(S,T)}$$

where the numerator denotes the normalization factor.

## 5 Clustering event traces

We adopted the agglomerative clustering (or hierarchical clustering) technique with minimum variance criteria [5] for our analysis. Agglomerative clustering works by initially placing each data item (here, an event trace) into a different cluster and iteratively combining clusters that are closest until we obtain a single cluster. Different criteria can be used in choosing the two clusters to combine in an iteration. We use the minimum variance criteria which tries to optimize the variance *within* a cluster. A detailed description of this approach is beyond the scope of this paper and the interested reader is referred to [3, 4].

**5.1 Evaluating the significance of clusters: A process mining perspective** Statistical metrics such as the average cluster density, silhouette width etc., have been proposed in the literature to evaluate the goodness of the clusters. The underlying motive for all these metrics is to prefer 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. The objective for clustering event logs is to ease the discovery of process models by grouping together traces that conform to similar execution patterns/behavior. To evaluate the significance of the clusters formed, one can compare the process models that are discovered from the traces within each cluster. In this paper, we propose two hypotheses to evaluate the goodness of clusters from a process mining point of view. Good clusters tend to cluster traces such that:

1. the process models mined show a high degree of fitness
2. the process models mined are less complex

The rationale behind these evaluation criteria is that if the clusters formed are meaningful (all traces belonging to related cases are in the same cluster and traces that are unrelated are not), then the process models resulting from the traces in each cluster should be less complex (more comprehensible and less spaghetti like). Algorithm 3 depicts the evaluation approach. Algorithm 3 is run over various clustering criteria/techniques and choice of cluster size.

**Algorithm 3** Evaluating the significance of clusters

**Require:** Given an event log $\mathcal{L}$ consisting of $M$ traces, and a clustering algorithm $\mathcal{C}$

**Ensure:** Partition the $M$ traces into $N$-clusters (for some $N \geq 2$) using $\mathcal{C}$

1: Discover the process model $P_i$ for each cluster, $C_i$, $1 \leq i \leq N$
2: Evaluate the fitness of the process models $P_i$
3: Evaluate the complexity of the process models. The number of control-flows, and/xor joins/splits and the size of the model defined in terms of the nodes, transitions and arcs signify the complexity of a process model.

The following clustering techniques are studied:

- A1: Bag-of-activities approach, Euclidean distance, agglomerative clustering
- A2: k-gram model, Euclidean distance, agglomerative clustering
- A3: Levenshtein distance, agglomerative clustering
- A4: Generic edit distance; substitution/indel scores as derived in Section 4, agglomerative clustering

## 6 Experimental Results and Discussion

We evaluate the above techniques using the telephone repair process event log (described in Section 4.1). There were a total of 1104 process instances in this data set. In addition to the whole data set, we have chosen random subsets of this data set for analysis. Subsets of 40%, 50%, 60%, 70%, 80%, 90% of instances have been

| Cluster No | A1 | | | | A2 | | | | A3 | | | | A4 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | $n_i$ | $f_i$ | $c_i$ | $s_i$ | $n_i$ | $f_i$ | $c_i$ | $s_i$ | $n_i$ | $f_i$ | $c_i$ | $s_i$ | $n_i$ | $f_i$ | $c_i$ | $s_i$ |
| 1 | 47 | .84 | 13 | 9 | 57 | **1.0** | 0 | 0 | 43 | .91 | 10 | 7 | 57 | **1.0** | 0 | 0 |
| 2 | 89 | .84 | 10 | 8 | 120 | .95 | 1 | 1 | 89 | .84 | 10 | 8 | 34 | **1.0** | 0 | 0 |
| 3 | 240 | .89 | 5 | 5 | 95 | .86 | 12 | 9 | 74 | .84 | 21 | 12 | 190 | **.91** | 12 | 9 |
| 4 | 31 | .82 | 21 | 12 | 138 | .89 | 12 | 8 | 127 | .88 | 5 | 5 | 52 | **.91** | 8 | 7 |
| 5 | 39 | .91 | 7 | 5 | 36 | .95 | 1 | 1 | 113 | **1.0** | 0 | 0 | 113 | **1.0** | 0 | 0 |

Table 4: Fitness and complexity metrics of the process models from the four clustering techniques for a random subset of 40% of the repair data set. The number of clusters is 5.

chosen randomly. For each such set of instances, we have applied the above clustering techniques. We have generated process models using the Alpha++ mining algorithm [8] (available in the ProM framework) over the traces in each cluster. The conformance checker plugin in ProM is used to measure the fitness of the process models thus generated. Further, we used the Petri-Net Complexity Analysis plugin in ProM over the process models. 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 in the process model. The larger the value of these metrics, the more complex is the model. The relationship between these metrics and comprehensibility has been reported in [9] while their relationship with defect/errors was reported in [10].

Table 4 depicts the fitness and complexity metrics of the process models mined from the traces clustered using the four techniques on 40% of the repair example data set. The data set has been partitioned into five clusters. In Table 4, $n_i$ signifies the number of instances in cluster $i$ while $f_i$ signifies the fitness of the process model mined using the instances of cluster $i$. $c_i$ signifies the number of control flows in the process model mined from instances of cluster $i$ while $s_i$ signifies the sum of and/xor joins/splits. It is interesting to note that the generic edit distance based clustering outperforms other techniques i.e., this technique is able to cluster the traces more coherently. The coherency is reflected in the fact that three process models with a fitness of 1.0 can be generated using this technique. Further, the comprehensibility of the process models is significantly better since these models have less control flows and and/xor join/splits. Clustering using Euclidean distance on k-grams (k is chosen to be 3) has the next best performance. This boosts the argument that incorporating context improves the goodness of clusters. Clustering using Euclidean distance on the bag-of-activities has the worst performance amongst the four while the Levenshtein distance based technique performs on par with the bag-of-activities.

We define two metrics viz., *average fitness* - $f_{\text{avg}}$ and *weighted average fitness* - $wf_{\text{avg}}$ as follows (here, $N$ is the number of clusters):

$$f_{\text{avg}} = \frac{1}{N}\sum_{i=1}^{N} f_i \qquad wf_{\text{avg}} = \sum_{i=1}^{N}(n_i * f_i)/\sum_{i=1}^{N} n_i$$

Weighted average fitness balances the fitness over imbalanced clusters[3]. For example, consider the scenario where 100 instances are partitioned into four clusters with $n_1 = 5$, $n_2 = 6$, $n_3 = 9$, $n_4 = 80$ instances. Assume that the process models mined from the first three clusters has a fitness value of 1.0 while the model from the fourth cluster has a fitness value of 0.8. The average fitness value would be $(1.0 + 1.0 + 1.0 + 0.8)/4 = .95$. However, the distribution of instances is skewed in the clusters and the average fitness value does not reflect the reality. The weighted average fitness value for this partitioning would be $(5 * 1.0 + 6 * 1.0 + 9 * 1.0 + 80 * 0.8)/100 = .84$, a realistic summarization of the goodness of clusters.

Table 5 illustrates the minimum, maximum and average fitness of the process models mined from the traces clustered using the four techniques on different subsets (varying between 40% and 100%) of the repair example data set. Each subset has been partitioned into five clusters. It is to be noted that the generic edit distance based technique performs consistently superior over the other techniques. The $k$-gram based approach which incorporates certain context is second best while the bag-of-activities based technique has the least fitness.

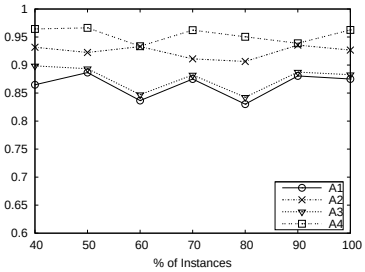
Figure 5 depicts the average and weighted average fitness of the process models mined from the traces clustered using the four techniques on different subsets of the telephone repair process log. It is to be noted that the generic edit distance based clustering

[3]The partitioning of instances is skewed in that the number of instances in some clusters are much less compared to others

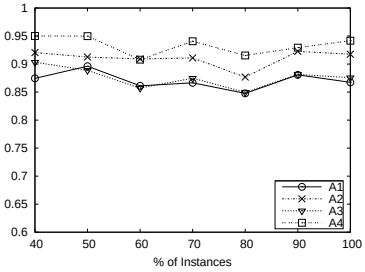
| Data Set Size | A1 | | | A2 | | | A3 | | | A4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| % | min | avg | max | min | avg | max | min | avg | max | min | avg | max |
| 40 | .82 | .86 | .91 | .85 | .93 | **1.0** | .84 | .89 | **1.0** | **.91** | **.96** | **1.0** |
| 50 | .82 | .88 | .92 | .88 | .92 | .95 | .84 | .89 | .95 | **.91** | **.96** | **1.0** |
| 60 | .80 | .83 | .89 | **.84** | **.93** | .95 | .80 | .84 | .95 | .80 | **.93** | **1.0** |
| 70 | .82 | .87 | .91 | .86 | .91 | .95 | .80 | .88 | .95 | **.89** | **.96** | **1.0** |
| 80 | .80 | .83 | .86 | .80 | .90 | .95 | .80 | .84 | .95 | **.83** | **.95** | **1.0** |
| 90 | .82 | .88 | .92 | .85 | **.93** | **1.0** | .82 | .88 | .95 | **.87** | **.93** | **1.0** |
| 100 | .82 | .87 | .92 | .86 | .92 | .95 | .80 | .88 | .95 | **.89** | **.96** | **1.0** |

Table 5: The minimum, average and maximum fitness values of the process models from the four clustering techniques for different subsets of the repair dataset. The number of clusters is 5.

yields better clusters over others techniques on both the metrics. Figure 6 depicts the complexity viz., average control flow and the total number of and/xor join/splits in the process models mined from the traces clustered using the four techniques on different subsets of the telephone repair process log. Again it can be observed that the context aware clustering techniques such as the generic edit distance and $k$-gram approach tend to generate process models that are less complex compared to the bag-of-activities approach.



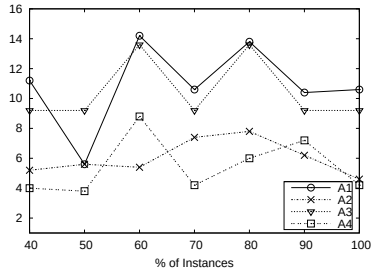

(a) Average Fitness



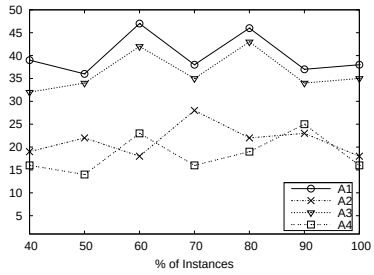

(b) Weighted Average Fitness

Figure 5: Average and weighted average fitness of the process models mined from the traces clustered using the four techniques on different subsets of the telephone repair process log

The results presented in this section are for the scenario where the data set is partitioned into 5 clusters. We have varied the number of clusters into which the data set is partitioned into and in all the cases a similar result was obtained.




(a) Average number of control flows




(b) Total number of and/xor join/splits

Figure 6: Complexity of process models mined from the traces clustered using the four techniques on different subsets of the telephone repair process log

In this study, we have used the Alpha++ mining algorithm to generate process models. However, other mining algorithms that yield process models amenable for analysis of the evaluation metrics defined in this paper can be used. Similarly, one can use other clustering techniques [4] instead of the agglomerative hierarchical clustering.

**6.1 Computational Complexity** The vector space approaches with Euclidean distance has a linear time complexity with respect to the number of features. For distance between two traces, this amounts to $\mathcal{O}(|\mathcal{A}|)$

and $\mathcal{O}(|\mathcal{A}|^k)$ respectively for the bag-of-activities and $k$-gram approaches. Edit distance computation (both Levenshtein and generic edit distance) between two traces takes quadratic time.

## 7 Related Work

Data clustering is one of the most important fields of data mining and a lot of techniques exist in the literature [3], [4], [5]. There is a growing interest in process mining and many case studies have been performed to show the applicability of process mining e.g., [11]. The significance of trace clustering to process mining has been discussed in [6], [12]. Greco et al. [6] used trace clustering to partition the event log and this way discovered more simple process models. They used the vector space model over the activities and their transitions to make clusters. Transitions can be considered as a specific case of the $k$-gram model where the value of $k$ is 2. On similar lines, Song et al. [7] have proposed the idea of clustering traces by considering a combination of different perspectives of the traces (such as activities, transitions, data, performance etc) as the feature vector. For the activities and transition perspectives, this approach can be thought of as a combination of the bag-of-activities and the $k$-gram approach (with $k = 2$). Though this combined approach might yield better results than either of the approaches in isolation, it still suffers from the pitfalls highlighted in Section 3. The generic edit distance based approach proposed in this paper is shown to outperform the vector-space model on these two perspectives. Further more, the generic edit distance considers the entire trace in totality thereby preserving the complete context of the process instance. Distances on other perspectives (such as data, performance etc) can be seamlessly combined with the generic edit distance just like in [7]. This helps in further boosting the results of process mining algorithms by leveraging the superior performance of the generic edit distance. A comprehensive list of metrics that influence the comprehensibility of process models was reported in [9].

## 8 Conclusions and Future Directions

In this paper, we have proposed a generic edit distance based approach to trace clustering. In order to tackle the sensitivity of the cost function (of edit operations) in the generic edit distance framework, we proposed an algorithm that automatically derives the cost of edit operations. The costs derived using this approach are shown to be effective. Further, we have proposed a process mining perspective to evaluate the goodness of clusters. It was shown that the proposed clustering approach outperforms contemporary approaches to trace clustering in process mining. The Alpha++ mining algorithm and conformance checker have been used to evaluate the goodness of clusters. However, there is a bias associated with a mining algorithm over the class of process models that it can generate and thereby the evaluation metrics. So far, little research has been done in this area. As future work, we would like to investigate the influence (bias) of a mining algorithm on the evaluation criteria.

**Acknowledgments** The authors are grateful to Philips Healthcare for funding the research in Process Mining.

## References

[1] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster, *Workflow Mining: Discovering Process Models from Event Logs*, IEEE Trans. Knowl. Data Eng., 16(9) (2004), pp. 1128-1142.

[2] E. S. Ristad and P. N. Yianilos, *Learning String-Edit Distance*, IEEE Trans. PAMI., 20-5 (1998), pp. 522-532.

[3] A. K. Jain, M. N. Murty, and P. J. Flynn, *Data Clustering: A Review*, ACM Computing Surveys, 31-3 (1999), pp. 264-323.

[4] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[5] J.H. Ward, *Hierarchical Grouping to Optimize an Objective Function*, J. Amer. Stat. Assoc., 58 (1963), pp. 236-244.

[6] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca, *Discovering Expressive Process Models by Clustering Log Traces*, IEEE Trans. Knowl. Data Eng., (2006), pp. 1010-1027.

[7] M. Song, C.W. Gunther, and W.M.P. van der Aalst *Trace Clustering in Process Mining*, BPM Workshops (2008) (to appear)

[8] L. Wen, W.M.P. van der Aalst, J. Wang, and J. Sun, *Mining Process Models with Non-Free Choice Constructs*, Data Min. Knowl. Discov., 15-2 (2007), pp. 145-180.

[9] J. Mendling, and M. Strembeck, *Influence Factors of Understanding Business Process Models*, BIS (2008), pp. 142-153.

[10] J. Mendling, G. Neumann, and W.M.P. van der Aalst, *Understanding the Occurrence of Errors in Process Models Based on Metrics*, OTM Conferences 1 (2007), pp. 113-130.

[11] W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. *Business Process Mining: An Industrial Application*, Info. Sys., 32-5: (2007), pp. 713-732.

[12] A.K. Alves de Medeiros, A. Guzzo, G. Greco, W.M.P. van der Aalst, A.J.M.M. Weijters, B.F. van Dongen, and D. Sacca, *Process Mining Based on Clustering: A Quest for Precision*, BPM Workshops (2007), pp. 17-29.