View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UPCommons. Portal del coneixement obert de la UPC

A High-level Strategy for C-net Discovery

Marc Solé and Josep Carmona
Software Department
Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain
msole@ac.upc.edu, jcarmona@lsi.upc.edu

***Abstract*—Causal nets have been recently proposed as a suitable model for process mining, due to their declarative semantics and compact representation. However, the discovery of causal nets from a log is a complex problem. The current algorithmic support for the discovery of causal nets comprises either fast but inaccurate methods (compromising quality), or accurate algorithms that are computational demanding, thus limiting the size of the inputs they can process. In this paper a high-level strategy is presented, which uses appropriate clustering techniques to split the log into pieces, and benefits from the additive nature of causal nets. This allows amalgamating structurally the discovered Causal net of each piece to derive a valuable model. The claims in this paper are accompanied with experimental results showing the significance of the high-level strategy presented.**

***Keywords*-Process discovery, Causal nets, High-level strategy, Clustering;**

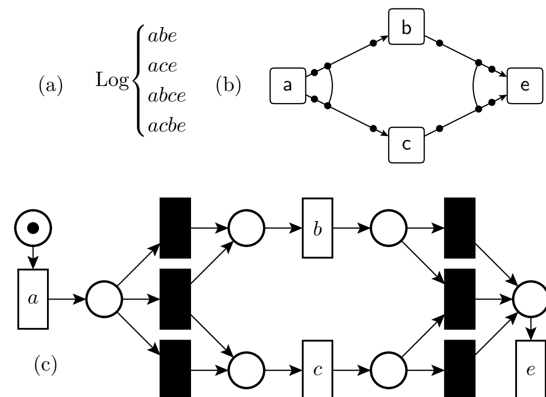

Figure 1. (a) A log. (b) Causal net describing the log. (c) Single Petri net modeling the same log with the help of silent transitions.

I. Introduction

The continuous growth of data generated by information systems has originated the advent of *Process Mining*, a discipline that sits in between the data mining and software engineering fields. The data, often available as a set of traces (*logs*) that information systems generate, can be processed in order to *discover* formal process models, check whether the available models *conform* to the reality observed in the log, and *extend* the current processes for improving the quality of the information provided [1].

There are several open problems in the area of Process Mining. A crucial one is *Control Flow Process Discovery*: given a log, to find a formal model (*e.g.*, a Petri Net) for a process which: i) represents (most of) the causal relations between activities in the log, ii) conforms [2] with a high degree the log, and iii) its graphical description is as much structured as possible [3]. Obviously, the selection of a given target model (Transition Systems, Petri nets, Heuristic nets, among others) influences the type of algorithms one can use for discovery. Moreover, given a formal model, there exist many possibilities depending on the target subclass. For instance the complexity of the α-algorithm [4] to discover *workflow nets* is very low when compared to the theory of regions [5], [6], [7], [8], which allows to discover general Petri nets.

Recently, a formalism called Causal nets (C-nets) [9] has been proposed as a suitable modeling language for process mining. It is a rather compact representation that allows expressing complex behavior that it is sometimes difficult to describe using other formalisms. For instance consider the log in Fig. 1(a). In Fig. 1(c) a Petri net that contains all the sequences in the log is described. In order to represent exactly the log (*i.e.*, to avoid incorporating in the net extra behavior), the use of *silent* transitions is required, graphically represented as black boxes. On the other hand, a simple C-net representing the log is shown in (b), which is quite compact. The semantics of that C-net can be informally described as:

Activity a *must be executed initially, since no obligations (input arcs with dots) exist for* a*. It can generate obligations to either 1) activity* b*, or 2) activity* c *or 3) activities* b *and* c*. Any of the these three possibilities requires the execution of the corresponding activities, consuming the obligation(s) from activity* a *and generating obligation(s) to activity* e*. The final execution of* e *will empty the set of obligations and therefore will lead to a valid trace.*

The problem of C-nets discovery poses new challenges that previous algorithms for discovery of other models like Petri nets did not have to address. First, the declarative (or a-posteriori) semantics of a C-net is defined on *valid firing sequences* (sequences that result in an empty set of pending

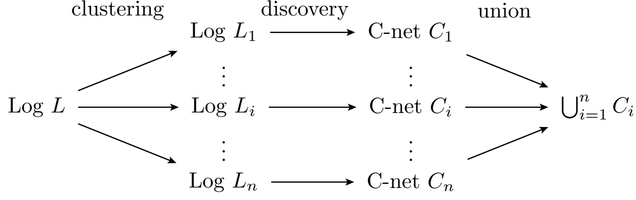

Figure 2. Proposed discovery workflow.

obligations), and therefore, C-nets are not prefix-closed. Second, contrary to Petri nets that have a restrictive nature (*i.e.*, the addition of a place can only restrict the behavior), C-nets have an *additive* nature, that is, the addition of elements to the C-net can only add behavior. For instance, by adding an arc with two dots between a and e in Fig. 1(b), the sequence ae is added to the language of the C-net.

This paper describes a high-level strategy to discover a C-net from a log. Although there are currently few discovery algorithms for C-nets in the literature, we expect their number to increase in the near future. The available discovery algorithms can be roughly classified in two classes: either they are fast and can handle large inputs [10] but may provide unsatisfactory results (for instance the C-nets generated by [10] can deadlock) or they provide high-quality C-nets (in terms of i), ii) and iii) above) but they are computationally expensive [11].

The additive feature of C-nets makes them specially suitable for the divide-and-conquer strategy presented in this paper (see Fig. 2). It is based on structurally combining several C-nets that result on applying discovery methods to small fractions of the log. The partitioning of the log is done by tailored clustering algorithms which can be guided by particular factors (frequency, similarity, balance, time, among others). Since the fractions tend to be small, they are tractable for high-quality discovery algorithms like [11]. Importantly, the union of the set of discovered C-nets can be accomplished structurally, a crucial fact that makes C-nets a very suitable model. If the same approach would have been done using a different model like Petri nets, one has to face the problem of deriving a Petri net which is the union of a set of Petri nets, an important issue that cannot be tackled structurally in general (a state-based solution to this problem which is grounded on the theory of regions can be found in [12]).

The organization of the paper is as follows: in Sect. II we introduce some mathematical notation and the formal definition of C-nets. Section III explains the properties of the union of C-nets. These properties are then used in Sect. IV to propose a high-level strategy to discover accurate C-nets from large inputs. The strategy is then experimentally tested on Sect. V. Section Sect. VI incorporates a discussion on the connection of this work with existing contributions in the literature, while Sect. VII concludes this paper and provides some research directions for the future.

II. Background

A. Mathematical preliminaries

A multiset (or a bag) is a set in which elements of a set X can appear more than once, formally defined as a function $X \to \mathbb{N}$. We denote as $\mathbb{B}(X)$ the space of all multisets that can be created using the elements of X. Let $M_1, M_2 \in \mathbb{B}(X)$, we consider the following operations on multisets: sum $(M_1 + M_2)(x) = M_1(x) + M_2(x)$, subtraction $(M_1 - M_2)(x) = \max(0, M_1(x) - M_2(x))$ and inclusion $(M_1 \subseteq M_2) \Leftrightarrow \forall x \in X, M_1(x) \leq M_2(x)$.

A log L is a bag of sequences of activities. In this work we restrict the type of sequences that can form a log. In particular, we assume that all the sequences start with the same initial activity and end with the same final activity, and that these two special activities only appear once in every sequence. This assumption is without loss of generality, since any log can be easily converted to satisfy these requirements by using two new activities that are properly inserted in each trace.

Given a finite sequence of elements $\sigma = e_1 e_2 \ldots e_n$, its length is denoted $|\sigma| = n$. The alphabet of σ, denoted A_σ, is the set of elements in σ. We extend this notation to logs, so that A_L is the alphabet of the log L, *i.e.*, $A_L = \bigcup_{\sigma \in L} A_\sigma$. Finally, the number of occurrences of a given element e in σ, *i.e.* $|\{e_i \mid e_i = e\}|$, is denoted as $\#(\sigma, e)$.

B. Causal nets (C-nets)

In this section we introduce the main model used along this paper.

Definition 1 (Causal net [9]): A Causal net is a tuple $C = \langle A, a_s, a_e, I, O \rangle$, where A is a finite set of activities, $a_s \in A$ is the start activity, a_e is the end activity, and I (and O) are the set of possible input (output resp.) bindings per activity. Formally, both I and O are functions $A \to S_A$, where $S_A = \{X \subseteq \mathcal{P}(A) \mid X = \{\emptyset\} \vee \emptyset \notin X\}$, and satisfy the following conditions:

- $\{a_s\} = \{a \mid I(a) = \{\emptyset\}\}$ and $\{a_e\} = \{a \mid O(a) = \{\emptyset\}\}$
- all the activities in the graph $(A, \mathrm{arcs}(C))$ are on a path from a_s to a_e, where $\mathrm{arcs}(C)$ is the *dependency relation induced by I and O* such that $\mathrm{arcs}(C) = \{(a_1, a_2) \mid a_1 \in \bigcup_{X \in I(a_2)} X \wedge a_2 \in \bigcup_{Y \in O(a_1)} Y\}$.

Definition 1 slightly differs from the original one from [9], where the set $\mathrm{arcs}(C)$ is explicitly defined in the tuple. The C-net of Fig. 1(b) is formally defined as $C = \langle \{a, b, c, e\}, a, e, I, O \rangle$, with $I(a) = \emptyset$, $O(a) = \{\{b\}, \{c\}, \{b, c\}\}$, $I(b) = \{\{a\}\}$, $O(b) = \{\{e\}\}$, $I(c) = \{\{a\}\}$, $O(c) = \{\{e\}\}$, $I(e) = \{\{b\}, \{c\}, \{b, c\}\}$ and $O(e) = \emptyset$. The dependency relation of C, which corresponds graphically to the arcs in the figure, in this case is $\mathrm{arcs}(C) = \{(a, b), (a, c), (b, e), (c, e)\}$. The activity bindings are denoted in the figure as dots in the arcs, *e.g.*,

$\{b\} \in O(a)$ is represented by the dot in the arc (a, b) that is next to activity a, while $\{a\} \in I(b)$ is the dot in arc (a, b) next to b. Non-singleton activity bindings are represented by arcs connecting the dots: $\{b, c\} \in O(a)$ is represented by the two dots in arcs (a, b), (a, c) that are connected through an arc.

Definition 2 (Binding, Binding Sequence, Projection): Given a C-net $\langle A, a_s, a_e, I, O\rangle$, the set B of activity bindings is $\{(a, S^I, S^O) \mid a \in A \ \wedge \ S^I \in I(a) \ \wedge \ S^O \in O(a)\}$. A binding sequence $\beta \in B^*$ is a sequence of activity bindings. By removing the input and output bindings from a binding sequence β, we do obtain an activity sequence denoted as σ_β.
Two binding sequences of the C-net in Fig. 1(b) are: $\beta^1 = (a, \emptyset, \{b\})(b, \{a\}, \{e\})(e, \{b\}, \emptyset)$ and $\beta^2 = (a, \emptyset, \{b, c\})(c, \{a\}, \{e\})(e, \{c\}, \emptyset)$. The projection of β^1 is $\sigma_{\beta^1} = abe$.

The semantics of a C-net are based on characterizing, among all the binding sequences it has, those ones that satisfy certain properties and therefore their corresponding projection (see Def. 2) will belong to the language of the C-net. The next definition addresses this.

Definition 3 (State, Valid Binding Sequence, Language): Given a C-net $C = \langle A, a_s, a_e, I, O\rangle$, its state space $S = \mathbb{B}(A \times A)$ is composed of states that represent multisets of pending obligations. Function $\psi \in B^* \to S$ is defined inductively: $\psi(\epsilon) = \emptyset$ and $\psi(\beta \cdot (a, S^I, S^O)) = \psi(\beta) - (S^I \times \{a\}) + (\{a\} \times S^O)$. The state $\psi(\beta)$ is the state of the C-net after the sequence of bindings β. The binding sequence $\beta = (a_1, S^I_1, S^O_1) \ldots (a_n, S^I_n, S^O_n)$ is said to be *valid* if:

1) $a_1 = a_s$, $a_n = a_e$ and $\forall k : 1 < k < n, a_k \in A \setminus \{a_s, a_e\}$
2) $\forall k : 1 \leq k \leq n, (S^I_k \times \{a_k\}) \subseteq \psi(\beta_{k-1})$
3) $\psi(\beta) = \emptyset$

The set of all valid binding sequences of C is denoted as $V_{\text{CN}}(C)$. The language of C, denoted $\mathcal{L}(C)$, is the set of activity sequences that correspond to a valid binding sequence of C, *i.e.*, $\mathcal{L}(C) = \{\sigma_\beta \mid \beta \in V_{\text{CN}}(C)\}$.
For instance, in Fig. 1(b), β^1 is a valid binding sequence, while β^2 is not, since the final state is not empty (condition 3 is violated). The language of that C-net is $\{abe, ace, abce, acbe\}$.

Importantly, C-nets can naturally represent behavior that cannot be easily expressed in the Petri net notation unless unobservable (silent) transitions are used. Fig. 3 illustrates this point. In the C-net, activities c and d can occur concurrently or exclusively, even in different iterations of the loop created by activity f, *e.g.*, $abcdez$ or $abdefbcez$. However, there is still another possibility that arises from combining the AND-split and the XOR-join: $abdceefbdfbcez$. Note that in this last trace, activity e could execute twice for a single b (although in the overall trace they execute the same number of times).

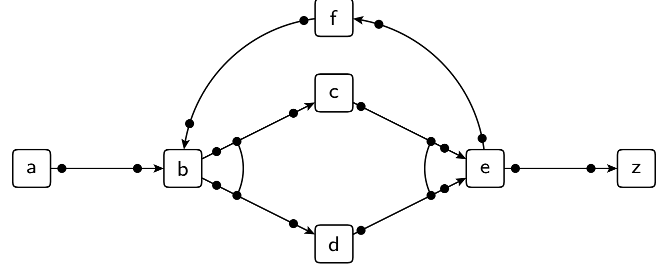

Figure 3. C-net mixing concurrent and exclusive behavior for activities c and d.

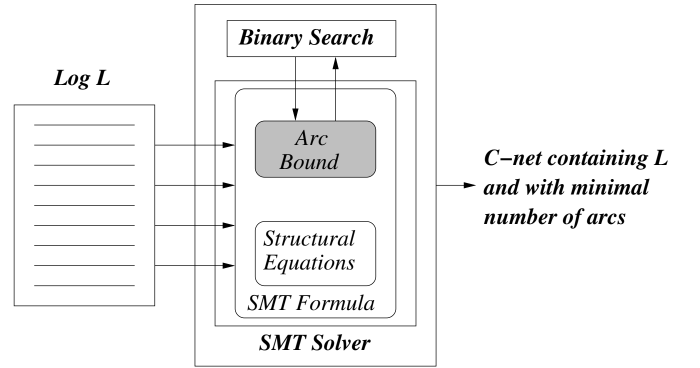

Figure 4. SMT technique for C-net discovery.

C. C-net discovery using SMT

We informally describe the strategy to derive a C-net from a log based on Satisfiability Modulo Theories (SMT), presented in [11]. The approach is shown in Fig. 4. First, the log is used to construct an SMT formula representing the possible bindings that each activity can have in a potential C-net that includes as valid sequences any trace in the log. Then the formula is augmented with an upper bound on the number of arcs the derived C-net can have, which can also be codified in the domain of SMT with the theory of quantifier-free bit-vector arithmetic [13]. This upper bound can be initially computed by counting the arcs of a C-net that is built using simple ordering relation between activities and whose language is guaranteed to contain all the sequences in the log (see the formal details in [11]).

On the other hand, a simple connectivity criteria can be used to also derive a simple lower bound, by using the alphabet of the log A_L: $|A_L| - 1$. Then, if an upper and lower bound on the number of arcs of the derived C-net are available, a binary search strategy can be used to seek for the minimal C-net that both includes the language of the log and has the minimal number of arcs. The approach iteratively invokes an SMT solver to determine whether the current

arc bound used does not harm satisfiability of the formula. Hence, based on the outcome of the SMT solver, the binary search strategy will update the bounds accordingly.

The method in [11] guarantees that i) the traces in the log are included in the set of valid binding sequences of the derived C-net (see Def. 2), *i.e.* the model derived is *fitting* [2], and ii) it has the minimal number of arcs. To the best of our knowledge, there is no other technique in the literature that either guarantees fitting models or limits the number of arcs. Hence, this will be the discovery technique used in our high-level strategy.

III. C-NET UNION

C-nets, contrary to Petri nets, have an "additive" nature. That is, while adding a place to a Petri net can only restrict behavior, adding an arc (or any other element) to a C-net can only add behavior. The "additive" nature of C-nets is formally defined with the help of Def. 4 and Property 1.

Definition 4: Given two C-nets $C_1 = \langle A_1, a_s^1, a_e^1, I_1, O_1 \rangle$ and $C_2 = \langle A_2, a_s^2, a_e^2, I_2, O_2 \rangle$, we say that C_1 is included in C_2, denoted $C_1 \subseteq C_2$, if:

- $a_s^1 = a_s^2 \wedge a_e^1 = a_e^2$,
- $A_1 \subseteq A_2$, and
- $\forall a \in A_1, I_1(a) \subseteq I_2(a) \wedge O_1(a) \subseteq O_2(a)$

For instance the C-net C_1 of Fig. 5(a) is included in the C-net of Fig. 5(c), but it is not included in the C-net of Fig. 1(b).

Property 1: Let C_1 and C_2 be two C-nets. If $C_1 \subseteq C_2$, then $V_{\text{CN}}(C_1) \subseteq V_{\text{CN}}(C_2)$, $\mathcal{L}(C_1) \subseteq \mathcal{L}(C_2)$.

Proof: Since the input and output binding sets of C_2 include the input and output binding sets of both C_1 (Def. 4), any valid binding sequence in C_1 will also be a valid binding sequence of C_2, thus $V_{\text{CN}}(C_1) \subseteq V_{\text{CN}}(C_2)$ which entails $\mathcal{L}(C_1) \subseteq \mathcal{L}(C_2)$ because the language of a C-net is obtained by simply keeping only the sequences of activities executed in the binding sequences. ■

As C_1 of Fig. 5(a) is included in Fig. 5(c), its language, $\{abce\}$, is a subset of the language of Fig. 5(c). Note that the union does not necessarily give the smallest C-net (in terms of its corresponding language) that can contain the union of languages of the united C-nets. For instance, the C-net in Fig. 1(b) includes both $\mathcal{L}(C_1)$ and $\mathcal{L}(C_2)$ but its language is a proper subset of the C-net of Fig. 5(c).

In spite of the fact that minimality of behavior is not guaranteed by the union operator (as the previous example demonstrates), still Property 1 makes the union of C-nets a very simple and effective operation to generate C-nets that include the behavior of the united C-nets.

Definition 5: Given two C-nets with identical initial and final activities, $C_1 = \langle A, a_s, a_e, I, O \rangle$ and $C_2 = \langle A', a_s, a_e, I', O' \rangle$, their union, denoted $C_1 \cup C_2$, is the C-net

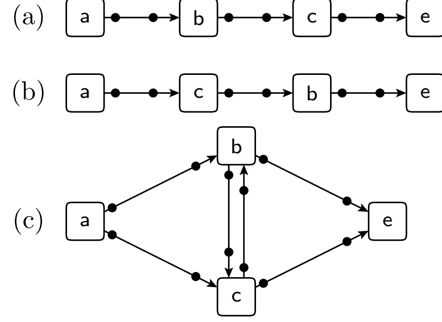

Figure 5. (a) C-net C_1 with language $abce$. (b) C-net C_2 with language $acbe$. (c) Union C-net $C_1 \cup C_2$. Its language (using regular expressions) is $ab(cb)^*e \cup a(bc)^+e \cup ac(bc)^*e \cup a(cb)^+e$.

$\langle A \cup A', a_s, a_e, I'', O'' \rangle$, where

$$I''(a) = \begin{cases} I(a) \cup I'(a) & \text{if } a \in A \cap A' \\ I(a) & \text{if } a \in A \setminus A' \\ I'(a) & \text{otherwise.} \end{cases}$$

$$O''(a) = \begin{cases} O(a) \cup O'(a) & \text{if } a \in A \cap A' \\ O(a) & \text{if } a \in A \setminus A' \\ O'(a) & \text{otherwise.} \end{cases}$$

For instance C-net C_1 and C_2 of Fig. 5(a) and (b), have the same initial and final activities, thus they can be united. Their union is the C-net of Fig. 5(c).

Lemma 1: Given two C-nets C_1 and C_2, $V_{\text{CN}}(C_1 \cup C_2) \supseteq V_{\text{CN}}(C_1) \cup V_{\text{CN}}(C_2)$, thus $\mathcal{L}(C_1 \cup C_2) \supseteq \mathcal{L}(C_1) \cup \mathcal{L}(C_2)$.

Proof: Since by Def. 5 $C_1 \subseteq C_1 \cup C_2$ and $C_2 \subseteq C_1 \cup C_2$, by Property 1 we know $V_{\text{CN}}(C_1) \subseteq V_{\text{CN}}(C_1 \cup C_2)$ and $V_{\text{CN}}(C_2) \subseteq V_{\text{CN}}(C_1 \cup C_2)$, thus $V_{\text{CN}}(C_1 \cup C_2) \supseteq V_{\text{CN}}(C_1) \cup V_{\text{CN}}(C_2)$. ■

The union will be a crucial operator in the approach presented in this paper, enabling the splitting of a log into pieces and amalgamating the individual results by using the union of C-nets. The next section is devoted to present this high-level strategy.

IV. A DIVIDE-AND-CONQUER STRATEGY

A. A clustering algorithm for C-nets

Clustering is an active research area in process discovery and there are numerous sequence clustering algorithms available (see Sect. VI for a discussion on related work). The approaches are very diverse, ranging from algorithms that assign to the same cluster sequences that are near when they are mapped to an n-dimensional space, to approaches that rely on sequence alignment techniques. Any of these techniques could be used in the clustering phase of our strategy. However, none of them is tailored to produce clusters particularly suitable for C-net discovery. In this section we propose a simple clustering method that tries to take advantage of the C-net specificities.

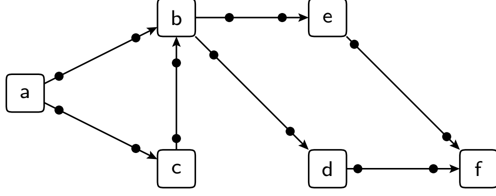

Figure 6. C-net with optional behavior (activity c) and a choice between d and e.

Ideally we would like to avoid situations like the following: consider the log $\{abce, acbe\}$. If we partition this log into two singleton clusters, namely $\{abce\}$ and $\{acbe\}$, and the simplest C-net for each cluster is generated, we obtain the C-nets C_1 and C_2 of Fig. 5(a) and Fig. 5(b), respectively. The union of these two C-nets is shown in Fig. 5(c). Clearly the C-net of Fig. 5(c) contains many additional behavior in the language of the net with respect to the original log. In general, this phenomenon will occur when particular orderings of concurrent behavior are assigned to different clusters.

On the other hand, let us consider the log $L = \{abdf, abef, acbdf, acbef\}$, which can be represented by the C-net of Fig. 6. This log exhibits typical constructs like choice (between activities d and e) and optional behavior (activity c). In this case, creating four singleton clusters, computing their simplest C-nets and merging them will yield precisely the C-net of Fig. 6, thus including no additional behavior. This is due to the fact that C-nets represent choice and optional behavior with additional arcs. Note, nevertheless, that in general clustering traces in this way may also introduce additional behavior: for instance consider the same log L but now without the sequence $acbef$, let us call this log L'. The result of splitting L' into three singleton clusters and merging the corresponding C-nets will yield exactly the same C-net as before, thus sequence $acbef$ will still belong to the language of the net, while it is possible to construct a C-net whose language is only L' (excluding $acbef$). However, note also that the simplest C-net for L' is again the C-net of Fig. 6, since larger input/output binding sets (involving additional arcs) are required to exclude sequence $acbef$ from the language.

Considering this particularity of C-nets, we propose the recursive clustering method shown in Algorithm 1.

The algorithm is invoked with two parameters, the log L and a threshold t, and will return the set of clustered logs that either have less that t sequences or could not be further split. First of all (line 2), it computes the set of activities A_s that could be used to partition L. These are the activities that both appear in at least one sequence of L, and do not appear in at least one sequence of L. Then it checks if the stop conditions are met (lines 3-5). In such a case, it simply returns the singleton set containing L. Otherwise, it

Algorithm 1 C-net oriented clustering

1: **function** RECURSIVESPLIT(L,t)
2: $A_s \leftarrow \{a \in A_L \mid \exists \sigma \in L : a \notin A_\sigma\}$
3: **if** $(|L| < t) \vee (A_s = \emptyset)$ **then**
4: **return** $\{L\}$
5: **end if**
6: $a \leftarrow$ selectSplitActivity(A_s, L)
7: $S_1 \leftarrow$ recursiveSplit$(\{\sigma \in L \mid a \in A_\sigma\}, t)$
8: $S_2 \leftarrow$ recursiveSplit$(\{\sigma \in L \mid a \notin A_\sigma\}, t)$
9: **return** $S_1 \cup S_2$
10: **end function**

performs the recursive part of the algorithm, by selecting first one activity among the set of candidate activities A_s (line 6). The are several possible heuristics to decide which candidate is better. In our current implementation the `selectSplitActivity` function simply chooses the activity that will yield more balanced partitions. Finally, two clusters are formed by considering the sequences in which the selected activity is present or not, and the function is called recursively on them.

With respect to our initial objective of clustering together concurrent behavior, this issue is fairly related with the concept of *synchronic distance* [14], which provides a degree of mutual dependence between two activities. In our setting, the synchronic distance of activities a_1 and a_2 in a log L can be formally defined as the maximal value of

$$|\#(\sigma', a_1) - \#(\sigma', a_2)|$$

where $\sigma = \sigma' \cdot \gamma$ is a trace of L. When two activities are totally independent, like it happens when activities are in conflict or one of them is optional, their synchronic distance is large (it can be as large as the length of a trace). On the other hand, concurrent or causality-related activities often have an small synchronic distance. Notice that selecting activities that appear in one trace and do not appear in another trace (as it is done in Algorithm 1), is a light approximation to select these activities with large synchronic distance, and hence candidates to have low dependency with at least some other activity in the log. Thus they are good candidates for splitting.

On the other hand, two concurrent activities are likely to have low synchronic distance, and therefore they will both appear in the sequence or none of them will appear. If one of them is selected as splitting activity, then with high probability the clusters that will arise from the splitting will keep the synchronic distance between these activities. However, in general as the recursive splitting progresses the synchronic distance between concurrent activities may change in the smaller logs generated, and therefore it is not advisable to perform too much recursive splitting. This is the reason to require in Algorithm 1 the minimal size a cluster must have.

B. A flexible divide-and-conquer strategy

The naive approach to the discovery phase of the methodology would be to simply run the discovery algorithm on each cluster. However by doing so we may miss important optimization opportunities. Since these tuning strategies are very dependent on the actual discovery algorithm used, let us center the discussion on the discovery algorithm proposed in [11]. This discovery algorithm basically translates the discovery problem to a Satisfiablity Modulo Theories (SMT) formula, and then uses an SMT solver to obtain a solution which can be converted into a C-net (see Sect. II-C).

Using such a strategy, it is possible to guarantee that all the sequences in each cluster will belong to the language of the corresponding C-net generated, and that there is no other C-net with less arcs that includes the cluster. Let us call `discoverMinArcCnet` the function that, given a cluster L_i, returns the C-net C_i with the previous properties. Algorithm 2 shows the divide-and-conquer strategy built around this function.

Algorithm 2 Divide-and-conquer independent strategy

1: **function** DIVIDEANDCONQUER(L,t)
2: $\{L_1, \ldots, L_n\} \leftarrow$ clusterLog(L, t)
3: **for** $L_i \in \{L_1, \ldots, L_n\}$ **do**
4: $C_i \leftarrow$ discoverMinArcCnet(L_i)
5: **end for**
6: $\{C'_1, \ldots, C'_m\} \leftarrow$ selectCnets$(\{C_1, \ldots, C_n\})$
7: **return** $\bigcup_{i=1}^{m} C'_i$
8: **end function**

First of all, the algorithm receives two parameters: the log L and the threshold t that determines the size of the sequence clusters generated by the clustering algorithm. The algorithm precisely starts with the clustering phase, with the call to the `clusterLog` function. We have deliberately avoided the direct use of the `recursiveSplit` function (Algorithm 1) to reinforce the idea that any clustering algorithm (or combination) can be used. In order to test the benefits of the clustering technique presented in the previous section, in our experiments we have used two clustering alternatives:

- The `recursiveSplit` function from Algorithm 1.
- A *random balanced clustering*, obtained by dividing the initial log into a given number of fragments of the same size (with exception of the last one that could have less traces).

For each cluster L_i in the set $\{L_1, \ldots, L_n\}$ of computed clusters, the function `discoverMinArcCnet` is called, which produces a C-net C_i whose language includes L_i and has the minimum number of arcs. Note that this step can be trivially parallelized and timeouts can also be set so that a C-net is generated in a given maximum amount of time (by uniting the C-nets of the clusters whose processing finished before the timeout, thus yielding also a fault-tolerant approach to process discovery), trading fitness (*i.e.*, capability for replaying the sequences in the log) for speed.

Finally, from the set of generated C-nets, a subset is selected (line 6). Here additional quality heuristics can be introduced. For instance, it is possible to select only the best C-nets in terms of the ratio: covered sequences per C-net arc, or the subset of C-nets whose union yields the best ratio, etc. In general, to guarantee that all the sequences belong to the language of the final C-net, all the generated C-nets must be selected. However, it is sometimes not adequate to use the whole set of C-nets for the union, since other factors may be more important than fitness. Among the possible factors to consider, there are two which are often contemplated.

- *Noise*: when a high percentage of noise is detected in a cluster, it may be advisable to not use the corresponding C-net. Noise can be sometimes detected with traditional data mining techniques [15].
- *Readability*: this is a subjective factor that may be estimated on the graph structure of the C-net.

C. An arc minimizing divide-and-conquer strategy

Algorithm 2 offers a flexible framework to adapt the C-net discovery algorithms to different requirements. However, in its most straightforward implementation (*i.e.*, using `recursiveSplit` or the *random balanced clustering* as the `clusterLog` function and selecting all the C-nets in the `selectCnets` function), the final C-net computed does not necessarily have the minimum number of arcs, although each one of its component C-nets has. The reason is that all the minimizations are local to each cluster and do not take into account the arcs of the C-nets found by applying the discovery technique to the other clusters.

A possible scheme to alleviate this inconvenient is shown in Algorithm 3. The basic idea in this case is to sort the clusters according to some criterion (*e.g.*, by the number of sequences they represent, by the size of their alphabet, etc.)

Algorithm 3 Divide-and-conquer incremental strategy

1: **function** INCDIVIDEANDCONQUER(L,t,α)
2: $(L_1, \ldots, L_n) \leftarrow$ sortLogs(clusterLog(L, t))
3: $C_1 \leftarrow$ discoverMinArcCnet(L_1)
4: $C \leftarrow C_1$
5: $b \leftarrow \max(|A_L| - 1, |\text{arcs}(C)|)$
6: **for** $L_i \in (L_2, \ldots, L_n)$ **do**
7: lb $\leftarrow \left| A_{L_i} \setminus \bigcup_{j=1}^{i-1} A_{L_j} \right| + 1$
8: ub $\leftarrow \alpha \cdot b - |\text{arcs}(C)|$
9: $C_i \leftarrow$ discoverMinNewArcCnet$(L_i, C, \text{lb}, \text{ub})$
10: $C \leftarrow C \cup C_i$
11: **end for**
12: **return** C
13: **end function**

and then discover the smallest C-net for the first cluster in the order, *i.e.*, L_1. Then, for the remaining clusters $L_2, \ldots, L_n$, instead of searching for the C-net with the minimal amount of arcs, the objective will be to minimize the number of additional arcs (with respect to the current C-net C). This task is performed by function `discoverMinNewArcCnet`, which is quite easy to implement in the discovery algorithm of [11], since we must simply remove from the part of the SMT formula that bounds the number of arcs (see Fig. 4 in Sect. II-C) the arcs already in C. This will minimize the number of new arcs introduced.

Remember that the discovery algorithm of [11] is able to minimize the number of arcs by performing a binary search, thus we have to provide some new bounds when minimizing the number of additional arcs. Lines 5, 7 and 8 tackle this. Let us consider first the lower bound in line 7. This is computed as the difference between the alphabet of the current cluster L_i and all previous clusters. By Def. 1 any C-net activity (besides a_s and a_e, that, in any case, cannot belong to this set difference because they always appear in all clusters) have at least one outgoing and one incoming arc. Thus the smallest structure in which k new activities can be inserted in a C-net, requires at least $k+1$ arcs (for instance connecting all k activities in a row, and then connecting the endings of the row to some activities already in the net).

For the upper bound, a similar theoretical limit can be found. However, it is more practical to define a user upper bound on the relative number of arcs that the user wants. This parameter, named α in the algorithm, indicates the fraction of additional arcs allowed, being $\alpha \cdot b$ the total number of arcs that the final C-net can have. The value for the b variable is computed as the maximum between the minimum number of arcs that a C-net with alphabet A_L can have ($|A_L| - 1$) or the number of arcs found in the first cluster. In case there is no C-net in the given bounds, the `discoverMinNewArcCnet` function returns the empty C-net, which is the neutral element with respect to C-net union.

V. EXPERIMENTS

The main purpose of this section is to illustrate the benefits of using the high-level technique presented in this paper with respect to the monolithic application of the technique in [11]. For that, we have selected a small set of benchmarks for which the aforementioned technique has problems to tackle, and we provide here the results for these benchmarks using the techniques described in this paper. Also, we compare the clustering technique proposed in Algorithm 1 with the naive strategy of random balanced clustering, described informally in Section IV-B.

Table I shows some basic information on the logs used in our experiments. The first five benchmarks are well-known logs from [6]. They were selected because they are difficult to tackle with the monolithic approach of [11]. For the sake of readability, the names of the logs have been shortened by removing "f0n00" before the underscore, *i.e.* `a22_5` refers to `a22f0n00_5`. The last one log is the `msweb` benchmark, which is a selection of 15000 sequences from the *Microsoft Anonymous Web Data* database in the UCI KDD repository[1]. The table contains the following information for each benchmark: $|L|$ is the number of (non-distinct) sequences in the log, $|L_u|$ is the number of distinct sequences, $|\sigma|$ is the length of the largest sequence and $|A_L|$ is the size of the alphabet of activities.

Log	$\lvert L\rvert$	$\lvert L_u\rvert$	$\lvert\sigma\rvert$	$\lvert A_L\rvert$
a22_5	900	836	76	22
t32_1	200	100	360	33
a32_5	900	900	102	32
a42_1	100	100	58	42
a42_5	900	900	78	42
msweb	15000	4250	20	236

Table I
BENCHMARK INFORMATION.

Tables II and III give information on the execution of each one of the clustering algorithms explained in Sect. IV-A: t is the threshold value used to partitioning the log, *cl* is the number of clusters produced. For each benchmark, we report particular information on each one of the clusters provided in columns L_1 to L_8 (the maximum number of clusters achieved in the benchmarks is 8). For each cluster, the number of sequences, number of distinct sequences (if different), and the size of the alphabet is shown. Focusing on this table, the first impression is the good balance in size for the derived clusters, both in number of traces and size of the log. This is a desirable feature of any divide-and-conquer technique, enabling a significant reduction both in peak memory consumption and CPU time.

Tables IV and V show the results of the divide-and-conquer strategy of Algorithm 3 on the benchmark logs compared with the monolithic approach of [11], using for clustering Algorithm 1 and the random balanced clustering, respectively. For the cluster initial sorting of Algorithm 3 we chose no particular ordering, thus clusters were sequentially processed (first L_1, then L_2, etc.). The tables contain the following information: *arcs* is the number of arcs of the final C-net, CPU is the elapsed time (in seconds) required to complete the discovery process, α is the α parameter of Algorithm 3, *cl* indicates the fraction of clusters successfully processed. For these experiments we have limited the maximum amount of memory to be used to 1Gb, and the maximum amount of elapsed time to one hour. The expressions *mem* and *time* in the CPU columns indicate which of both limits was reached. The α parameter was set in each case to a value that allowed us to unite all the resulting C-nets.

[1] Available at: http://kdd.ics.uci.edu/databases/msweb/msweb.html

Log	t	cl		L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8
a22_5	150	8	$\|L\|$	103	103	119	129	109	86	133	118
			$\|L_u\|$	93	103	101	128	96	85	112	118
			$\|A_L\|$	18	19	16	21	18	19	16	21
t32_1	50	3	$\|L\|$	90	26	84	–	–	–	–	–
			$\|L_u\|$	45	13	42	–	–	–	–	–
			$\|A_L\|$	32	32	33	–	–	–	–	–
a32_5	150	8	$\|L\|$	92	138	90	135	113	120	110	102
			$\|A_L\|$	20	26	20	26	30	31	25	31
a42_1	50	3	$\|L\|$	23	28	49	–	–	–	–	–
			$\|A_L\|$	35	41	41	–	–	–	–	–
a42_5	150	8	$\|L\|$	110	119	109	112	112	115	114	109
			$\|A_L\|$	31	36	32	37	33	39	40	40
msweb	1000	7	$\|L\|$	3739	760	3343	2249	3635	641	633	–
			$\|L_u\|$	926	371	566	854	766	311	456	–
			$\|A_L\|$	188	142	171	165	182	135	153	–

Table II
RESULTS OF THE RECURSIVESPLIT CLUSTERING FUNCTION (ALGORITHM 1).

Log	t	cl		L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8
a22_5	113	8	$\|L\|$	113	113	113	113	113	113	113	109
			$\|L_u\|$	111	111	111	112	112	113	111	106
			$\|A_L\|$	22	22	22	22	22	22	22	22
t32_1	67	3	$\|L\|$	67	67	66	–	–	–	–	–
			$\|L_u\|$	67	67	66	–	–	–	–	–
			$\|A_L\|$	33	33	33	–	–	–	–	–
a32_5	113	8	$\|L\|$	113	113	113	113	113	113	113	109
			$\|A_L\|$	32	32	32	32	32	32	32	32
a42_1	34	3	$\|L\|$	34	34	32	–	–	–	–	–
			$\|A_L\|$	42	42	42	–	–	–	–	–
a42_5	113	8	$\|L\|$	113	113	113	113	113	113	113	109
			$\|A_L\|$	42	42	42	42	42	42	42	42
msweb	2143	7	$\|L\|$	2143	2143	2143	2143	2143	2143	2142	–
			$\|L_u\|$	828	800	793	818	845	843	807	–
			$\|A_L\|$	167	177	190	172	187	171	174	–

Table III
RESULTS OF A BALANCED RANDOM CLUSTERING.

To report on the quality of the derived C-nets, Tables IV and V additionally have the last three columns. For each benchmark, these three values are meant to estimate two main features of the model derived: i) *fitness* [2] and ii) similarity with the model derived using the monolithic approach. Informally, in the fitness dimension the capability of the log in replaying log traces is measured. Column *cf* is the *cost-based fitness per case* metric of [16], where 1.0 indicates that all sequences in the log belong to the language of the C-net, and the smaller the value is, the less sequences are reproducible by the C-net.

A metric complementary to fitness is *precision*: it is used to determine the amount of extra behavior that the model contains but was not observed in the log [2], [17]. Since the precision measures for C-nets are not yet developed, there is no way to quantify the amount of additional behavior of the generated model with respect to the C-net of the monolithic approach. For this reason we have defined two similarity measures to compare two C-nets generated for

	monolithic [11]			d&c-rsplit (Sect. IV)						
Log	arcs	CPU	cf	cl	α	arcs	CPU	cf	as	iobs
a22_5	34	265	1.0	8/8	1.5	34	50	1.00	1.00	0.998
t32_1	–	mem	–	2/3	1.5	45	253	0.97	–	–
a32_5	46	814	1.0	8/8	2.0	46	114	1.00	1.00	0.977
a42_1	63	248	1.0	3/3	1.5	67	53	1.00	0.94	0.940
a42_5	–	mem	–	8/8	1.7	65	519	1.00	–	–
msweb	–	time	–	7/7	2.0	1079	441	1.00	–	–

Table IV
RESULTS OF THE DIVIDE-AND-CONQUER STRATEGY COMPARED TO THE MONOLITHIC APPROACH.

	d&c-rand (Sect. IV)						
Log	cl	α	arcs	CPU	cf	as	iobs
a22_5	8/8	1.0	34	24	1.00	1.00	1.000
t32_1	0/3	1.1	–	–	–	–	–
a32_5	8/8	1.0	46	62	1.00	1.00	0.977
a42_1	3/3	1.1	63	57	1.00	1.00	0.984
a42_5	8/8	1.1	64	719	1.00	–	–
msweb	7/7	2.5	1282	632	1.00	–	–

Table V
RESULTS OF THE DIVIDE-AND-CONQUER STRATEGY USING A RANDOM BALANCED CLUSTERING.

the same log. These two similarity measures, namely *arc similarity* and *input/output binding set similarity*, appear in columns *as* and *iobs* in Table IV, respectively. Given two C-nets $C_1 = \langle A, a_s, a_e, I_1, O_1 \rangle$ and $C_2 = \langle A, a_s, a_e, I_2, O_2 \rangle$, these similarity measures are defined as follows:

$$\text{as}(C_1, C_2) = \frac{|\text{arcs}(C_1) \cap \text{arcs}(C_2)|}{|\text{arcs}(C_1) \cup \text{arcs}(C_2)|}$$

$$\text{iobs}(C_1, C_2) = \left(\sum_{a \in A \setminus \{a_s\}} \frac{|I_1(a) \cap I_2(a)|}{|I_1(a) \cup I_2(a)|} + \sum_{a \in A \setminus \{a_e\}} \frac{|O_1(a) \cap O_2(a)|}{|O_1(a) \cup O_2(a)|} \right) \cdot \frac{1}{2 \cdot (|A| - 1)}.$$

These are normalized similarity measures, where values range between 1.0 (identical) to 0.0 (completely different). In particular the input/output binding set similarity is a more accurate indicator than the arc similarity in the sense that $\text{iobs}(C_1, C_2) = 1.0$ entails that C_1 is equal to C_2, while $\text{as}(C_1, C_2) = 1.0$ does not. For instance the C-nets C_1 of Fig. 1(b) and C_2 of Fig. 7(a) have $\text{as}(C_1, C_2) = 1.0$ but $\text{iobs}(C_1, C_2) = \frac{6}{10}$. Note also that low values do not necessarily imply that both C-nets have a different language, specially if the nets contain redundant bindings. For instance the C-nets of Fig. 5(a) and Fig. 7(b) have the same language, *i.e.*, $abce$, but their similarity values are very low (0.5 for the *as* metric, and $\frac{1}{3}$ for *iobs*).

The general conclusion that can be drawn from the experimental results of Table IV is the capability of the approach in handling benchmarks for which the monolithic

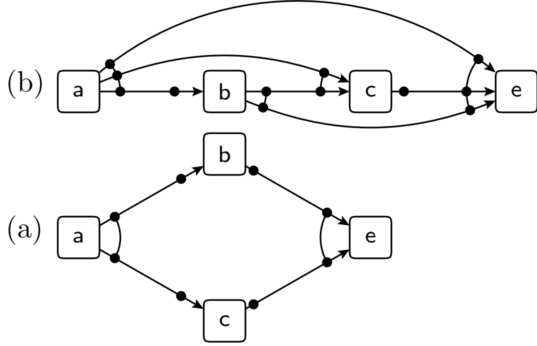

Figure 7. (a) C-net with arc similarity equal to 1.0 with C-net of Fig. 1(b). (b) C-net with low similarities with C-net of Fig. 5(a) but the same language.

technique of [11] fails. Moreover, for those benchmarks were the monolithic approach succeeds, the reduction in CPU time is considerable (roughly a x6 reduction). This reduction in time comes with no significant penalty in the quality of the derived C-nets: the *as* and *iobs* metrics computed for these C-nets are close to 1.0, which means that since the C-net derived for the monolithic approach is an optimal model in terms of number of arcs, then the approach of this paper is capable in finding models close to the optimal.

It is also worthwhile to compare the results of Tables IV and V, to have some insight on the benefits of the new clustering algorithm proposed in this paper. The naive approach of a random balanced clustering proved to be a valid method for most of the benchmarks but `t32_1`, which could not be handled due to memory problems in two of the three clusters, and CPU time expired in the remaining cluster. Comparing the running times between the two alternatives, they are on the same order of magnitude, but with differences sometimes reaching an x2 factor.

One of the elements that helps in explaining these differences is the α factor used in each case. In general, the random balanced clustering strategy requires smaller values, which has a direct impact on the number of SMT problems that have to be solved. On the other hand, since this clustering is not specifically tailored for SMT, the time taken by each one of these iterations is usually larger than with the new clustering.

This faster processing of the SMT problems when Algorithm 1 is used for clustering can be explained by looking at the information of Tables IV and V. First of all, given a SMT formula, the number of variables and equation it contains are good predictors of the running time taken to decide its satisfiability. Since it was shown in [11] that these quantities are proportional to the number of distinct sequences ($|L_u|$), the size of the alphabet ($|A_L|$) and the length of the sequences, it is easy to see from these tables that clusters produced by Algorithm 1 are more easily processed than the ones created by the random strategy. On the other hand, the fact that all activities are present in every cluster produced by the random strategy (notably with the exception of `msweb`) tends to generate an initial number of arcs that is closer to the final one, thus allowing to reduce the α parameter (except, again, in the `msweb` log). All in all the clustering proposed effectively reduces the burden on the SMT solver, which has a major impact for instance while processing the `t32_1` log, but ways in which the global number of iterations could be decreased should be investigated.

VI. Related Work

The techniques presented in this paper are grounded on the traditional idea of divide and conquer a complex problem, by splitting the input into several pieces and combining the individual solutions into the final output. We will focus on the particular application of this strategy in the area of process discovery.

To the best of our knowledge, [18] was the first attempt to tackle process discovery by using clustering. The approach was extended further in [19]. For the clustering technique presented in [19], an abstraction is computed as a preprocess: each trace is projected into the most relevant features (computed previously) and associated with a vector of values. Then the *k-means* algorithm is used to partition the vectorial space defined by the traces. A similar strategy to this one was presented in [20].

A totally different approach is described in [21], [22], where the problem of multiple trace alignment is considered. These approaches are based on combining *trace alignment* techniques together with a *hierarchical clustering* algorithm. Although their complexity may be high, the use of alignment methods may be a crucial element to improve the quality of current process mining techniques [1].

Finally, in [23] an approach that is based on recursively projecting the traces of the log is presented. The partitioning of the log is not *horizontal* (*e.g.*, selecting traces of the log) like the one of this paper, but *vertical* (*e.g.*, selecting events of the log and projecting every trace onto these events). This allows ending up with traces that are amenable for process discovery since the final relations between events tend to be simpler.

VII. Conclusions and Future Work

This paper presents a high-level framework for the problem of C-net discovery. By using an specially tailored clustering algorithm, and adapting a monolithic C-net miner that relies on SMT, this paper demonstrates that the approach is able to handle inputs for which the monolithic application of the SMT-based discovery will fail. Remarkably, the approach is not sacrificing fitness, and can be guided to derive nets whose size is below some threshold. Moreover, when comparing the quality of the derived C-nets with respect to

the monolithic approach, there are no significant differences. The techniques of this paper are implemented in a tool, and experimental results on process discovery benchmarks demonstrate the feasibility of the contribution proposed to tackle these logs in limited time.

As future work, we plan to test the techniques of this paper on larger benchmarks. Moreover, the techniques of this paper can be extended in some dimensions. First, further extensions of the clustering algorithm will be explored, to enable guiding the discovery problem with particular objectives (*e.g.* readability, precision, handle noise, among others). Second, the problem of *C-net simulation* will be explored. This may allow for instance to alleviate the complexity of the clustering approach presented in this paper: each time a C-net is derived from a given cluster, remove from the remaining clusters the traces that are included by this C-net. Finally, other strategies to combine the C-nets obtained in the clusters which consider additional information will be considered.

ACKNOWLEDGMENT

This work has been supported by projects FORMALISM (TIN2007-66523) and TIN2011-22484.

REFERENCES

[1] W. M. P. van der Aalst, *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[2] A. Rozinat and W. M. P. van der Aalst, “Conformance checking of processes based on monitoring real behavior,” *Information and Systems*, vol. 33, no. 1, pp. 64–95, 2008.

[3] W. M. P. van der Aalst and C. W. Günther, “Finding structure in unstructured processes: The case for process mining,” in *ACSD*, T. Basten, G. Juhás, and S. K. Shukla, Eds. IEEE Computer Society, 2007, pp. 3–12.

[4] W. M. P. van der Aalst, T. Weijters, and L. Maruster, “Workflow mining: Discovering process models from event logs,” *IEEE TKDE*, vol. 16, no. 9, pp. 1128–1142, 2004.

[5] R. Bergenthum, J. Desel, R. Lorenz, and S.Mauser, “Process mining based on regions of languages,” in *Business Process Management*, Sep. 2007, pp. 375–383.

[6] J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik, “Process discovery using integer linear programming,” in *Petri Nets*, ser. LNCS, vol. 5062, 2008, pp. 368–387.

[7] J. Carmona, J. Cortadella, and M. Kishinevsky, “New region-based algorithms for deriving bounded petri nets,” *IEEE Trans. Computers*, vol. 59, no. 3, pp. 371–384, 2010.

[8] M. Solé and J. Carmona, “Process mining from a basis of state regions,” in *Petri Nets*, ser. LNCS, vol. 6128, 2010, pp. 226–245.

[9] W. Van Der Aalst, A. Adriansyah, and B. Van Dongen, “Causal nets: a modeling language tailored towards process discovery,” in *CONCUR*, 2011, pp. 28–42.

[10] A. J. M. M. Weijters and J. T. S. Ribeiro, “Flexible heuristics miner (FHM),” in *CIDM*. IEEE, 2011, pp. 310–317.

[11] M. Solé and J. Carmona, “An SMT-based discovery algorithm for C-nets,” (submitted to ATPN’12) also as tech. rep. at UPC, Tech. Rep. LSI-12-2-R, 2012. [Online]. Available: http://www.lsi.upc.edu/dept/techreps/techreps.html

[12] ——, “Incremental process mining,” *T. Petri Nets and Other Models of Concurrency*, 2012 (To Appear).

[13] S. Jha, R. Limaye, and S. Seshia, “Beaver: Engineering an efficient SMT solver for bit-vector arithmetic,” in *Computer Aided Verification*, 2009, pp. 668–674.

[14] T. Murata, “Petri Nets: Properties, analysis and applications,” *Proceedings of the IEEE*, pp. 541–580, Apr. 1989.

[15] L. Maruster, A. J. M. M. Weijters, W. M. P. van der Aalst, and A. van den Bosch, “A rule-based approach for process discovery: Dealing with noise and imbalance in process logs,” *Data Min. Knowl. Discov.*, vol. 13, no. 1, pp. 67–87, 2006.

[16] A. Adriansyah, B. van Dongen, and W. van der Aalst, “Conformance checking using cost-based fitness analysis,” in *Enterprise Distributed Object Computing Conference (EDOC)*, 2011, pp. 55 –64.

[17] J. Munoz-Gama and J. Carmona, “A Fresh Look at Precision in Process Conformance,” in *Business Process Management (BPM)*, 2010.

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

[19] A. K. A. de Medeiros, A. Guzzo, G. Greco, W. M. P. van der Aalst, A. J. M. M. Weijters, B. F. van Dongen, and D. Saccà, “Process mining based on clustering: A quest for precision,” in *Business Process Management Workshops*, ser. Lecture Notes in Computer Science, A. H. M. ter Hofstede, B. Benatallah, and H.-Y. Paik, Eds., vol. 4928. Springer, 2007, pp. 17–29.

[20] M. Song, C. W. Günther, and W. M. P. van der Aalst, “Trace clustering in process mining,” in *Business Process Management Workshops*, ser. Lecture Notes in Business Information Processing, D. Ardagna, M. Mecella, and J. Yang, Eds., vol. 17. Springer, 2008, pp. 109–120.

[21] R. P. J. C. Bose and W. M. P. van der Aalst, “Context aware trace clustering: Towards improving process mining results,” in *Proceedings of the SIAM International Conference on Data Mining, SDM 2009, April 30 - May 2, 2009, Sparks, Nevada, USA*. SIAM, 2009, pp. 401–412.

[22] ——, “Process diagnostics using trace alignment: Opportunities, issues, and challenges,” *Inf. Syst.*, vol. 37, no. 2, pp. 117–141, 2012.

[23] J. Carmona, “Projection approaches to process mining using region-based techniques,” *Data Min. Knowl. Discov.*, vol. 24, no. 1, pp. 218–246, 2012.