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

Amending C-net Discovery Algorithms (Technical Report)

Marc Solé[1,2] and Josep Carmona[2]

[1] Computer Architecture Department, UPC msole@ac.upc.edu
[2] Software Department, UPC jcarmona@lsi.upc.edu

Abstract. As the complexity of information systems evolves, there is a growing interest in defining suitable process models than can overcome the limitations of traditional formalisms like Petri nets or related. *Causal nets* may be one of such promising process models, since important characteristics of their semantics deviate from the ones in the literature. Due to their novelty, very few discovery algorithms exist for Causal nets. Moreover, the existing ones offer very few guarantees regarding the outcome produced. This paper describes an algorithm that can be applied as a second step to any discovery technique to significantly improve the quality of the final Causal net derived. We have tested the technique in combination with the existing algorithms in the literature on several benchmarks, noticing a considerable improvement in all of them.

1 Introduction

The discipline of *process discovery* addresses an important problem: to obtain a formal process model from a *log* (set of sequences of activities) [1]. There has been quite important progress in terms of process discovery algorithms in the last decade, but it is widely accepted that no *silver bullet* algorithm for process discovery has still been found, and therefore the investigation of *process enhancement* techniques for improving the quality of a model in the presence of a log may be crucial for the incorporation of process discovery as a common practice in industry. This paper presents a technique for enhancing the quality of a process model.

Causal nets (C-nets) [2] has been proposed as a suitable formalism for describing process models. Due to its compact representation, flexibility, and declarative semantics, complex behavior can be naturally expressed in a C-net. Fig. 1(b) shows an example of C-net. The informal semantics of this C-net is:

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 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.

There have been few atempts for C-net discovery in the literature. In [3–5], algorithms for discovering *heuristic nets* (a subclass of C-nets) are presented.

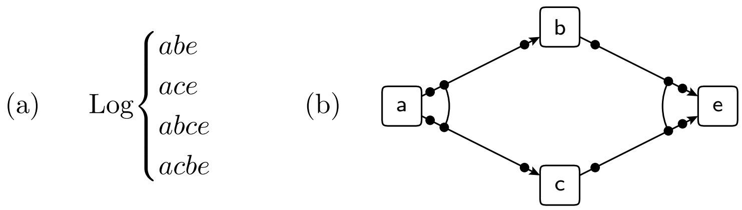

Fig. 1. (a) A log. (b) Causal net describing the log.

The more general approach among these three is [5], which discovers *flexible heuristic nets*. Unfortunately, these three approaches suffer from a fundamental problem: none of them can guarantee to derive *fitting* models [6], *i.e.*, in the worst case no sequence in the log is represented in the model.

To the best of our knowledge, the approach in [7] is the first algorithm for discovering the full class of C-nets. Remarkably, it offers two important guarantees: i) it derives a fitting model with ii) the minimal number of arcs. The approach uses a Satisfiability Modulo Theories (SMT) solver as the discovery engine. However, the use of such a complex machinery for discovery makes the approach rather limited for handling large inputs. A way to alleviate the complexity of this approach is to apply divide-and-conquer strategies on top of the SMT solver, as described in [8], using tailored clustering techniques to partition the log into small fractions that may be better handled by the solver. The combination of the individual C-nets obtained can be easily joined into the final C-net, due to the additive nature of the model. The experiments performed in [8] show the capability of the high-level strategy to handle larger inputs.

This paper presents a new application of the SMT-based approach from [7]. Instead of addressing discovery, it assumes an initial skeleton of a C-net (which can be obtained via discovery or manually), and *amends* the C-net in two dimensions: *fitness* and *simplicity* (see [1], Sect. 5.4.3). For fitness, it may incorporate new arcs and bindings necessary to reproduce the sequences in the log, while for simplicity it minimizes the number of arcs and bindings, removing the redundant ones. The core idea of the set of techniques of this paper strongly relies on adapting the SMT problem used for discovery in [7] to be used for simulation in this paper. One side-effect theoretical contribution of this paper is the reduction of the fitness decisional problem (*i.e.*, *does a C-net fit a log?*) into an SMT problem.

Organization of the paper: In Sect. 2 we give the formal definition of C-nets and we introduce some of the mathematical background used in the rest of the paper. Our approach to C-net enhancement is explained in Sect. 3 and experimentally tested in Sect. 4. Finally, Sect. 5 presents some future work and concludes this paper.

2 Background

2.1 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 $X \rightarrow \mathbb{N}$. 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 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 $|\sigma| = n$, and the element at position i (*e.g.*, e_i) is denoted σ_i. Its prefix sequence up to element i (but not including it), with $i \leq n+1$, denoted $\sigma_{\leftarrow i}$, is $e_1 \ldots e_{i-1}$. We define $\sigma_{\leftarrow 1}$ as the empty sequence, denoted ϵ. Conversely, its suffix sequence after i, with $i < n$, denoted $\sigma_{i \rightarrow}$, is $e_{i+1} \ldots e_n$. We express the fact that an element e appears in sequence σ as $e \in \sigma$. 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.*, $A_L = \bigcup_{\sigma \in L} A_\sigma$.

2.2 Causal nets (C-nets) and the discovery problem

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

Definition 1 (Causal net [2]). *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, $a_e \in A$ 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 $A \rightarrow S_A$, 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 [2], 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 $\{a\} \in I(a)$ is the dot in arc (a, b) next to b. Non-singleton activity bindings are represented by circular segments 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* $\mathrm{act}(\beta)$*.*

As example, two possible 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 $\mathrm{act}(\beta^1) = abe$.

The semantics of a C-net are achieved by selecting, among all the possible binding sequences, the ones satisfying certain properties. These sequences will form the set of *valid binding sequences* of the C-net, and their corresponding projection (see Def. 2) will define 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 $\beta = (a_1, S^I_1, S^O_1) \ldots (a_{|\beta|}, S^I_{|\beta|}, S^O_{|\beta|})$ is said to be* valid *if:*

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

The set of all valid binding sequences of C is denoted as $V(C)$. 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.*,* $\mathcal{L}(C) = \{\mathrm{act}(\beta) \mid \beta \in V(C)\}$.

For instance, for the C-net of Fig. 1(b), the sequence β^1 above 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 $\{abe, ace, abce, acbe\}$.

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 the following concepts. Given two C-nets C_1 and C_2 with the same initial and final activities, we say that C_1 is *included* in C_2, denoted $C_1 \subseteq C_2$, if all the input/output bindings of C_1 are also present in C_2. In such a case, then $V(C_1) \subseteq V(C_2)$, $\mathcal{L}(C_1) \subseteq \mathcal{L}(C_2)$ and $\mathrm{arcs}(C_1) \subseteq \mathrm{arcs}(C_2)$.

These properties make the union of C-nets an effective operation to generate C-nets that include the behavior of all the united C-nets. The *union* of two C-nets C_1 and C_2 with the same initial and final activities, denoted $C_1 \cup C_2$, is the C-net that contains all the activities and input/output bindings of C_1 and C_2.

Property 1. Given two C-nets C_1 and C_2, $\mathcal{L}(C_1 \cup C_2) \supseteq \mathcal{L}(C_1) \cup \mathcal{L}(C_2)$.

The C-net discovery problem is defined as follows: given a log L, find a C-net C such that $\mathcal{L}(C) \supseteq L$. Additionally, the derived C-net can be optimized in some dimension(s) (number of arcs, number of bindings, additional behavior, among others). Given the additive nature of C-nets, there is a simple method to generate a C-net that can replay all the traces in L. It is based on the *immediately follows* relation [9] between the activities in L, denoted $<_L$ and defined as $<_L = \{(a, b) \mid \exists \sigma \in L, \exists i : 1 \leq i < |\sigma| \ \wedge \ \sigma_i = a \ \wedge \ \sigma_{i+1} = b\}$.

Definition 4. *Given a log L, the* immediately follows C-net of *L, denoted $C_{IF}(L)$, is the C-net $\langle A, a_s, a_e, I, O\rangle$ such that: (i) $A = A_L$, (ii) $\forall \sigma \in L, \sigma_1 = a_s \wedge \sigma_{|\sigma|} = a_e$, (iii) $\forall a \in A,\ O(a) = \{\{b\} \mid a <_L b\} \ \wedge \ I(a) = \{\{b\} \mid b <_L a\}$.*

The immediately follows C-net can be computed in linear time with respect of the size of the log, but allows for many unobserved behavior, thus exhibiting a poor *precision* [6]. However, it is a simple alternative to generate a model for sequences in which other approaches have problems in dealing with.

2.3 C-net discovery using SMT

We briefly describe the strategy to derive a C-net from a log based on Satisfiability Modulo Theories (SMT), presented in [7]. The approach is shown in Fig. 2. 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.

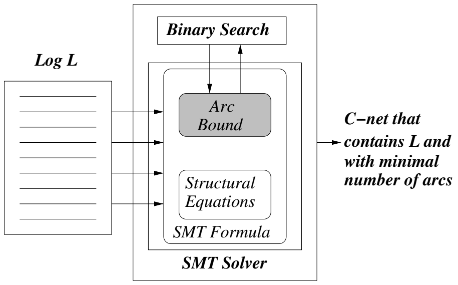

Fig. 2. SMT-based C-net discovery.

The construction of the formula from a log L is as follows. Given a sequence $\sigma \in L$, by using the equations (E1), (E2) and (E3) below, a binding sequence $\beta = (\sigma_1, X_1, Y_1) \ldots (\sigma_{|\sigma|}, X_{|\sigma|}, Y_{|\sigma|})$ is computed, such that $\text{act}(\beta) = \sigma$. The sets X_i and Y_i of β are encoded using integer variables over the domain $\{0, 1\}$ (*i.e.*, a Boolean variable, although we treat it as an integer in this section). In particular we use a variable $x_{\sigma,i,(a,\sigma_i)}$ to indicate whether activity a belongs to X_i in β ($x_{\sigma,i,(a,\sigma_i)} = 1$ indicates that the execution of activity σ_i *consumes* the obligation (a, σ_i)). Similarly, the variable $y_{\sigma,i,(\sigma_i,a)}$ encodes if a belongs to Y_i in β, so that if $y_{\sigma,i,(\sigma_i,a)} = 1$ then the execution of activity σ_i *produces* an obligation (σ_i, a). Using these variables, the following equations are built for σ:

(E1) Every activity (except the initial one) has to consume at least one obligation, *i.e.*, $X_1 = \emptyset$ and $X_i \neq \emptyset$ for $i > 1$. Formally, $\forall i : 1 < i \leq |\sigma|$, $\sum_{a \in A_{\sigma_{\leftarrow i}}} x_{\sigma,i,(a,\sigma_i)} \geq 1$.

(E2) Every activity (except the final one) has to produce at least one obligation, *i.e.*, $Y_{|\sigma|} = \emptyset$ and $Y_i \neq \emptyset$ for $i < |\sigma|$. Formally, $\forall i : 1 \leq i < |\sigma|$, $\sum_{a \in A_{\sigma_{i\rightarrow}}} y_{\sigma,i,(\sigma_i,a)} \geq 1$.

(E3) The state of obligations after executing the prefix $\beta_{\leftarrow i}$ (*i.e.*, $\psi(\beta_{\leftarrow i})$) contains, at least, the obligations in $(X_i \times \{\sigma_i\})$. This is the same as requiring that the number of obligations of the type (a, σ_i) in $\psi(\beta_{\leftarrow i})$ is larger or equal than the number of obligations (a, σ_i) in $(X_i \times \{\sigma_i\})$. Moreover, if i is the last occurrence of activity σ_i, then the previous relation must be an equality, since there cannot be pending obligations in the final state, *i.e.*, the last occurence of an activity must consume all the obligations for it. The number of such obligations in $\psi(\beta_{\leftarrow i})$ can be computed by summing the number of times the obligation has been produced minus the number of times it has been already consumed before the execution of σ_i. Since this equation is more involved than (E1) and (E2), please refer to [7] for the formal details.

Definition 5 (Structural equations, C-net of a satisfying assignment). *The set of* structural equations *for a C-net including the behavior of a log L, denoted* structural_equations(L)*, is the set of equations obtained by adding the equations (E1), (E2) and (E3) for every $\sigma \in L$. A satisfying assignment α of the structural equations gives a set of binding sequences B such that $\forall \sigma \in L, \exists \beta \in B : \mathrm{act}(\beta) = \sigma$. Then, the C-net of the satisfying assignment α is the C-net $C = \langle A, a_s, a_e, I, O\rangle$ with:*

- $A = \{a \mid \exists \beta \in B : (a, X, Y) \in \beta\}$
- $\forall a \in A, I(a) = \{X \mid \exists \beta \in B : (a, X, Y) \in \beta\}$
- $\forall a \in A, O(a) = \{Y \mid \exists \beta \in B : (a, X, Y) \in \beta\}$.

It was shown in [7] that any C-net C of a satisfying assignment to the structural equations satisfies $\mathcal{L}(C) \supseteq L$, and for any C-net C', if $\mathcal{L}(C') \supseteq L$ then C' includes a C-net corresponding to a satisfying assignment to the structural equations.

To minimize the number of arcs of the C-net, the formula obtained from the structural equations 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 [10]. This upper bound can be initially computed by counting the arcs of the immediately follows C-net of the log L. 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 [7] 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* [6], and ii) it has the minimal number of arcs.

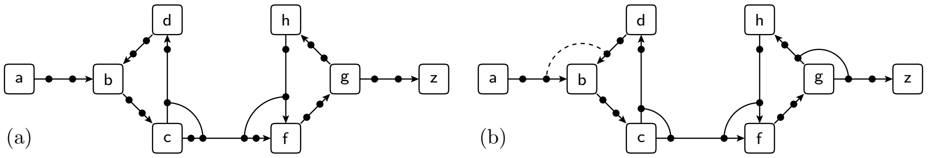

Fig. 3. Deadlock in a C-net discovered using the FHM plugin. The *refit* algorithm is able to generate (a) starting from (b).

3 C-net refitting

To improve the quality of a given C-net C with respect to a log L, we propose the *refit*(C,L) algorithm, that will be presented in Sect. 3.6. This algorithm is able, for instance, to derive the C-nets Fig. 3(a) and Fig. 4(a) starting from the C-nets of Fig. 3(b) and Fig. 4(b), respectively.

Once we informally illustrate the limitations of current C-net discovery algorithms (Sect. 3.1), the next sections present the necessary ingredients that will be used in the *refit*(C,L) algorithm. In Sect. 3.2 we show how to adapt the technique from [7] in order to alleviate the complexity of the discovery problem by using the skeleton of a given C-net. Then, in Sect. 3.3 the problem of simulating a log by a C-net is solved by extending the SMT model derived in Sect. 3.2. Two optimization techniques are presented in Sects. 3.4 and 3.5, that are ment to reduce the numbers of bindings. Sect. 3.6 will finally combine all these techniques for presenting the overall strategy of the *refit*(C,L) algorithm.

3.1 Limitations of current discovery methods

There are currently two approaches specifically devised for C-net discovery: the fast but heuristically based [5], and the slow and memory demanding but providing quality guarantees SMT-based [7, 8]. However, both approaches frequently produce non-optimal C-nets as next examples illustrate.

Our first example shows a usual drawback of the strategy in [5]. The C-nets produced by this strategy frequently have deadlocks and many (in the worst case, all) of the sequences in the log cannot be replayed in the model. For instance, we created a log by simulating the C-net in Fig. 3(a) and tried to rediscover it using the FHM plugin in ProM that implements the algorithm in [5]. However, since the approach is heuristic sometimes the concurrency or exclusive relations between activities are not correctly interpreted by the algorithm. In this case, the FHM plugin yields the C-net in Fig. 3(b). It is easy to see that this C-net deadlocks because of the input binding drawn with a dashed line, since for b to execute, activity d should have been executed, but d can only execute once b has. In this case to fix the net a more careful analysis of the relations between activities should have been performed.

On the other hand, although the strategy in [7] produces fitting C-nets with minimal arcs (see Sect. 2.3), these nets might contain redundant bindings. For

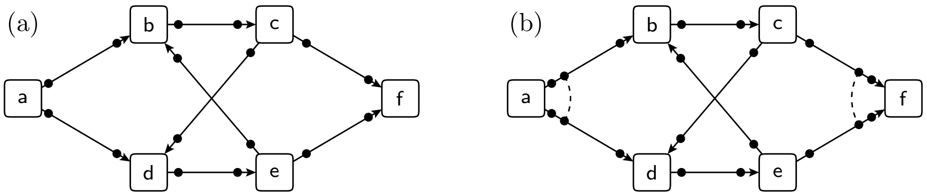

Fig. 4. Redundant bindings in a C-net discovered using an SMT-based approach. The *refit* algorithm is able to generate (a) starting from (b).

instance, consider the log $L = \{abcf, adef,\ abcdef, adebcf,\ abcdebcf\}$ which can be described by the C-net of Fig. 4(a). However, a possible output for the SMT-based approach is given in Fig. 4(b) which has two additional bindings, marked with dashed arcs. In particular, it allows the sequences of activities bc and de to be concurrent. This is a valid possibility in the sequences $abcdef$ and $adebcf$ of L, and, since the number of arcs is not affected by the inclusion of the additional bindings, then the algorithm does not prefer one C-net over the other, and the outcome is dependent on the SMT solver implementation. In this example, it is clear that the C-net in Fig. 4(b) can be improved by minimizing the number of bindings in the model. This paper proposes a way to do this, thus extending [7].

3.2 C-net discovery using a skeleton

The structural equations of a log (Def. 5) precisely describe the C-nets whose language includes the log. However, considering all this solution space might be prohibitive in terms of complexity for the SMT solver. For instance, in [7] the SMT solver could only complete for small logs when the whole set of equations was considered.

To alleviate the complexity of the problem, some simplifications are proposed in [7]. One possibility is to restrict the alphabets for which a given activity a can produce/consume an obligation, thus reducing the number of variables of the SMT problem. That is to substitute in the structural equations the sets $A_{\sigma_{i\rightarrow}}$ and $A_{\sigma_{\leftarrow i}}$ by some non-empty subset $A'_{\sigma_{i\rightarrow}}$ and $A'_{\sigma_{\leftarrow i}}$, respectively, such that $\emptyset \subset A'_{\sigma_{\leftarrow i}} \subseteq A_{\sigma_{i\rightarrow}}$ and $\emptyset \subset A'_{\sigma_{i\rightarrow}} \subseteq A_{\sigma_{\leftarrow i}}$. For instance, with this substitution equation (E1) becomes $\forall i : 1 < i \leq |\sigma|,\ \sum_{a \in A'_{\sigma_{\leftarrow i}}} x_{\sigma,i,(a,\sigma_i)} \geq 1$.

In [7] these reduced alphabets are computed by defining an activity window around every activity a: only activities that in some sequence are at most at a given distance from a are considered. This strategy enables the discovery of C-nets from larger logs, but still in some cases the simplifications are not enough.

The alternative considered in this paper is to use what we call a *C-net skeleton* to derive the reduced alphabets $A'_{\sigma_{i\rightarrow}}$ and $A'_{\sigma_{\leftarrow i}}$. The C-net skeleton is simply a C-net C whose arcs contain the relevant activity relationships, so that the arcs of the discovered C-net C' can only be a subset of the arcs of C, *i.e.* $\text{arcs}(C') \subseteq \text{arcs}(C)$. To achieve this restriction, the reduced alphabets are defined as $A'_{\sigma_{i\rightarrow}} = A_{\sigma_{i\rightarrow}} \cap \{a \mid (\sigma_i, a) \in \text{arcs}(C)\}$ and $A'_{\sigma_{\leftarrow i}} = A_{\sigma_{\leftarrow i}} \cap \{a \mid (a, \sigma_i) \in \text{arcs}(C)\}$. We denote as *skeleton_SMT(*C, L*)* the set of structural equations of a given log L when the activity sets are restricted using the arcs in C-net C.

Note that the arcs of the discovered C-net are fixed by the skeleton, but that the input/output bindings of C' are not restricted to the ones in C. This degree of flexibility is, precisely, what enables to include behavior in C' not previously accepted by C. This is specially important for skeletons of non-fitting C-nets, as will be demonstrated in Sect. 4.

3.3 Simulation as an SMT problem

The simulation of a log in a given C-net C can be expressed also as an SMT problem by using the structural equations of C (*i.e.*, skeleton_SMT(C, L)) plus a set of equations that restrict the possible values of the X and Y variables to the set of input and output bindings in C. Although alternative simulation methods are possible, this approach has some advantages. First of all, the straightforward C-net simulation of computing the state after the execution of each activity is worst-case exponential in the number of possible states[3]. Second, *the simulation problem of all the log can be solved in a single SMT problem instance* (which is NP-complete). Finally, the SMT-based simulation will be the basis for further optimizations as we will see in Sect. 3.4.

Formally, given a C-net $C = \langle A, a_s, a_e, I, O\rangle$ and a log L, for each sequence $\sigma \in L$ we add the following equations: $\bigwedge_{1<i\leq|\sigma|} \bigvee_{S\in I(\sigma_i)} X_i = S$ and $\bigwedge_{1\leq i<|\sigma|} \bigvee_{S\in O(\sigma_i)} Y_i = S$. Let us denote as *restrict_choices(*C, L*)* this set of additional equations.

Since the equations in restrict_choices(C, L) must be expressed using the Boolean variables in the X and Y sets (see Sect. 2.3), a number of issues have to be considered. For instance, the expression $X_i = S$ might be already false if $S \nsubseteq A'_{\sigma_{\leftarrow i}}$ which allows discarding some comparisons. For the remaining input bindings S, for which $S \subseteq A'_{\sigma_{\leftarrow i}}$, the expression $X_i = S$ is translated as $\bigwedge_{a\in S} x_{\sigma,i,(a,\sigma_i)} \wedge \bigwedge_{a\in A'_{\sigma_{\leftarrow i}}\setminus S} \overline{x_{\sigma,i,(a,\sigma_i)}}$. There is an analogous consideration for the Y sets.

We define the simulation SMT problem *simulate_SMT(*C, L*)* as:

$$\boxed{\text{simulate\_SMT}(C, L) \overset{\text{def}}{=} \text{skeleton\_SMT}(C, L) \wedge \text{restrict\_choices}(C, L)}$$

The solution to this SMT problem is the set of values of the X and Y variables from which the X_i and Y_i sets can be reconstructed. This means that from each sequence σ in the log, we can obtain a valid binding sequence β such that $\text{act}(\beta) = \sigma$.

[3] One can notice this with the simple example of Fig. 1(b): to simulate the occurrence of activity a, the three output bindings should be considered as potential successor states, in general to proceed with the simulation any of them can be combined with the occurrences of the sequent activities, which in turn may introduce new output binding possibilities.

Theorem 1. *The equations simulate_SMT(C, L) have a solution if, and only if, every sequence σ in L is replayable by C.*

Proof. $\boxed{\Rightarrow}$ We start by observing is that the equations in skeleton_SMT(C, L) are equivalent to structural_equations$(L) \wedge$ remove_arcs(C, L), where remove_arcs(C, L) is a formula that assigns zero to all X and Y variables that correspond to obligations of arcs not present in C (thus forbidding the presence of any arc not in C). Thus, simulate_SMT(C, L) is equivalent to structural_equations$(L) \wedge$ remove_arcs$(C, L) \wedge$ restrict_choices(C, L). Since by [7] we know that the solution to structural_equations(L) yields the valid binding sequences of a C-net C' such that $\mathcal{L}(C') \supseteq L$, and the possible solutions to simulate_SMT(C, L) are also solutions to structural_equations(L), then we know that from the solutions to simulate_SMT(C, L) we can obtain a C-net C' such that $\mathcal{L}(C') \supseteq L$. Now, because of remove_arcs$(C, L) \wedge$ restrict_choices(C, L), we know that $C \supseteq C'$ (for instance C might contain bindings never used to simulate the sequences in L) hence that $\mathcal{L}(C) \supseteq \mathcal{L}(C') \supseteq L$.

$\boxed{\Leftarrow}$ If every $\sigma \in L$ can be replayed by C, then there is a valid binding sequence β in $V(C)$ such that act$(\beta) = \sigma$. If simulate_SMT(C, L) has no solution this would entail that, since structural_equations(L) is always satisfiable (for every log there is at least one possible C-net including it), the contradiction is introduced either by remove_arcs(C, L) or restrict_choices(C, L). However the latter equations simply restrict the solution to use the elements of C, thus it contradicts the fact that $\beta \in V(C)$. □

3.4 Minimization of input/output bindings

The mechanism by which input and output bindings of a C-net can be minimized is closely related to the SMT-based simulation we have seen in the previous section. The basic idea is to build an SMT simulation problem in which we add an additional equation, enforcing that at least a given number of bindings are not used during the C-net simulation, *i.e.* the simulation problem then becomes *is it possible to simulate the net without using at least l of its bindings?*, where l is the desired number of unused bindings. Once we know how to establish this bound on the number of unused bindings, by performing a binary search we can maximize them, thus minimizing the number of required C-net bindings.

Formally, the quantity to maximize is:

$$\text{unused}(C, L) \overset{\text{def}}{=} \sum_{a \in A} \left(\left(\sum_{S \in I(a)} \bigwedge_{\sigma \in L} \bigwedge_{\sigma_i = a} X_i \neq S \right) + \left(\sum_{S \in O(a)} \bigwedge_{\sigma \in L} \bigwedge_{\sigma_i = a} Y_i \neq S \right) \right)$$

so given a (lower) limit l on the number of unused bindings, the SMT problem built is:

$$\boxed{\text{min\_unused}(C, L, l) \overset{\text{def}}{=} \text{simulate\_SMT}(C, L) \wedge (\text{unused}(C, L) \geq l)}$$

To be able to perform a binary search we must provide a range of possible values for the parameter l. The lower bound of this range is clearly zero, since it is possible that the C-net requires all its bindings. On the other hand, if C contains n bindings it is possible to give a tighter upper bound than simply n. First of all, any activity that is not the initial nor the final one, must have at least one input and one output bindings, while the initial (final) activity must have at least one output (input) binding. Thus if C contains $|A|$ activities, this means that at least $(|A|-2)\cdot 2+2 = 2\,|A|-2$ bindings are required, so $n-2\,|A|+2$ is a valid upper bound.

This upper bound can be improved with the information obtained during the creation of the formula $\text{unused}(C, L)$: if in any sequence some binding S is the only possible choice, then there is no point in keeping it in the formula, since it is mandatory and can never be unused. If an activity a has m mandatory input (resp. output) bindings the upper bound decreases by $m-1$ (since one input (output) binding per activity was already considered in the original bound).

3.5 Simplification of unfrequent bindings

To require perfect fitness is sometimes too stringent, since the log may contain noise or be incomplete [1]. Hence one may allow some percentage of the log to not be reproducible by the model, specially if the fraction left out is the responsible of complex constructs in the model whose corresponding behavior is not frequent in the log (this is known as the *80/20 model*).

The objective of the technique of this section is to remove as many bindings as possible while removing as few sequences of the log as possible from the language of the C-net. We assume the user provides a parameter f, which is the minimum amount of fitness that the simplified net must have. To solve this problem we follow a greedy strategy that iteratively selects the "best" binding to remove, a strategy that does not guarantee the best possible final selection of bindings, but works well in practice.

In particular, using the binding sequences provided by the simulation, each binding is annotated with the set of sequences where it appears. The binding that appears in the least number of sequences is selected, and the sequences where it appears form the set of removed sequences L_r. From that point on, we add the binding that introduces the least new sequences into L_r. In the case there are several bindings with the same amount of sequences (in the first iteration) or new sequences (in the remaining iterations), we consider the frequency of the set sequences: we select the binding that shares its set of (new) sequences with more other bindings. This way, we greedily maximize the number of removed bindings. We will see the impact of this greedy technique in the experiments performed in Sect. 4.

3.6 The algorithm

Algorithm 1 shows the *refit* algorithm. It starts by simulating all the sequences in the log L. Although we have seen in Sect. 3.3 an implementation of the *simulate* function that uses an SMT approach, any other function that decides whether a

Algorithm 1 C-net refitting

```
 1: function REFIT($C, L, f$)
 2:     if $\neg$simulate($C, L$) then                                      ▷ Sect. 3.3
 3:         $E \leftarrow$ skeleton_SMT($C, L$)                              ▷ Sect. 3.2
 4:         $feasible, solutions \leftarrow$ solve($E$)                ▷ Call SMT solver
 5:         if $\neg feasible$ then
 6:             $L_n \leftarrow \{\sigma \in L \mid \neg$simulate$(C, \{\sigma\})\}$   ▷ Non-replayable sequences
 7:             $C \leftarrow C \cup C_{\mathrm{IF}}(L_n)$                  ▷ Include missing behavior
 8:             $E \leftarrow$ skeleton_SMT($C, L$)
 9:             $feasible, solutions \leftarrow$ solve($E$)            ▷ Always solvable
10:         end if
11:         $C \leftarrow$ extract_cnet($solutions$)          ▷ Obtain C-net from SMT solution
12:         $C \leftarrow$ minimize_arcs($C, L$)                       ▷ Binary search of [7]
13:     end if
14:     $C \leftarrow$ minimize_bindings($C, L$)                              ▷ Sect. 3.4
15:     if $f < 1.0$ then
16:         $C \leftarrow$ simplify($C, L, f$)                                 ▷ Sect. 3.5
17:     end if
18:     return $C$
19: end function
```

given C-net C can replay the sequences in L could have been used, thus adding some flexibility to the approach.

C-nets that can replay all sequences in L are simply minimized in the number of bindings (line 14). This tends also to reduce the number of arcs as a side-effect[4]. Since the function that minimizes arcs (minimize_arcs) is not based on simulation as the function that minimizes bindings (minimize_bindings), it usually takes much more time to compute, and since the minimization of bindings also reduces the number of arcs we have chosen to skip the minimization of arcs in this case. Once the number of bindings has been reduced, if user demands a simplification of the unfrequent bindings of the net (in which case $f < 1.0$), then the *simplify* function Sect. 3.5 is executed. Otherwise the net is simply returned without any other modification.

On the other hand, if some sequence of L is not in the language of C, a first quick try to fix the net is performed by using the skeleton of C in an SMT-based approach. If the problem is solvable, then the C-net is computed and the procedure in [7] to minimize the number of arcs is applied to the net (line 12). Otherwise all non-replayable sequences are identified (line 6) and the union of the original C-net and the immediately follows C-net (c.f. Def. 4) is computed (line 7). The resulting C-net is guaranteed to include all the sequences in the log by Property 1, and it is used as the skeleton for another SMT problem. This time the problem is guaranteed to be solvable, thus a C-net is extracted from the solution (line 11) and the number of arcs minimized.

[4] In all our experiments the number of arcs obtained by the minimization of bindings was the same as the C-nets that were minimized in the number of arcs, see Sect. 4.

In summary, the algorithm presented can be used to combine any strategy for C-net discovery with an SMT-based approach to improve the quality (fitness, simplicity) of the derived model. Next section provides experiments on the combination of the *refit* algorithm with existing techniques for C-net discovery.

4 Experiments

The utility of the proposed *refit* algorithm (Sect. 3) has been tested in several scenarios. All the experiments have been run in an Intel Core Duo using the Linux 3.0 kernel in which the amount of memory used was limited to 1 Gb and the maximum allowed elapsed processing time was one hour. Table 1 shows the benchmarks used in the experiments. They are well-known logs from [12] and some small but complex examples from [7] (some of them obtained by simulation of C-nets in [2], namely the *at1* and *at2* logs). The table also includes some basic information like the number of distinct sequences in the log ($|L|$), the length of the largest sequence ($|\sigma_m|$) and the size of the alphabet of activities ($|A_L|$).

Log	Source	Original name	$\|L\|$	$\|\sigma_m\|$	$\|A_L\|$
at1	[7]	aalst1	10	5	5
at2		aalst2b	8	11	5
mxa		mixedXorAnd	3	14	7
opt		optional1	11	8	6
cyc		cycles	7	18	8
a15	[12]	a12f0n00_5	5	7	12
a21		a22f0n00_1	99	46	22
a31		a32f0n00_1	100	73	32
t31		t32f0n00_1	100	360	33
a41		a42f0n00_1	100	58	42
a25		a22f0n00_5	836	76	22
a35		a32f0n00_5	900	102	32
a45		a42f0n00_5	900	78	42

Table 1. Benchmarks used.

To test the refit algorithm, besides having the log, we need some starting C-nets. In this case we have used the three currently available algorithms that discover C-nets, that is: the monolithic SMT-based approach of [7], denoted *SMTmono*, the incremental SMT-based discovery approach of [8] (*SMTinc*) and the Flexible Heuristics Miner (*FHM*) plugin in ProM [5]. The C-nets generated by the refit algorithm from these nets have been labeled by adding *+refit* to the original generation method, so that *FHM+refit* refers to the C-net obtained by the refit algorithm from the C-net generated by the FHM plugin.

Fig. 5 contains a summary of the results contained in Tables 2, 3 and 4. The graph at the top compares the number of input/output bindings in the resulting net, the center graph shows the elapsed time to complete the execution of the corresponding algorithm (for the *+refit* methods, this includes the time to run the original discovery algorithm plus the time taken by the refit algorithm). Finally the bottom graph compares the quality of the nets using two quality measures: f is the ratio of sequences in the log that can be replayed by the model, while *cf* is the *cost-based fitness per case* metric of [13] 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 similar are the sequences of the log to the language of the C-net. We now provide the results of the two classes of techniques used in combination with the refit algorithm.

SMT-based techniques: We start by evaluating the capacity of the refit algorithm to minimize the number of input/output bindings (Fig. 5(top)). This part

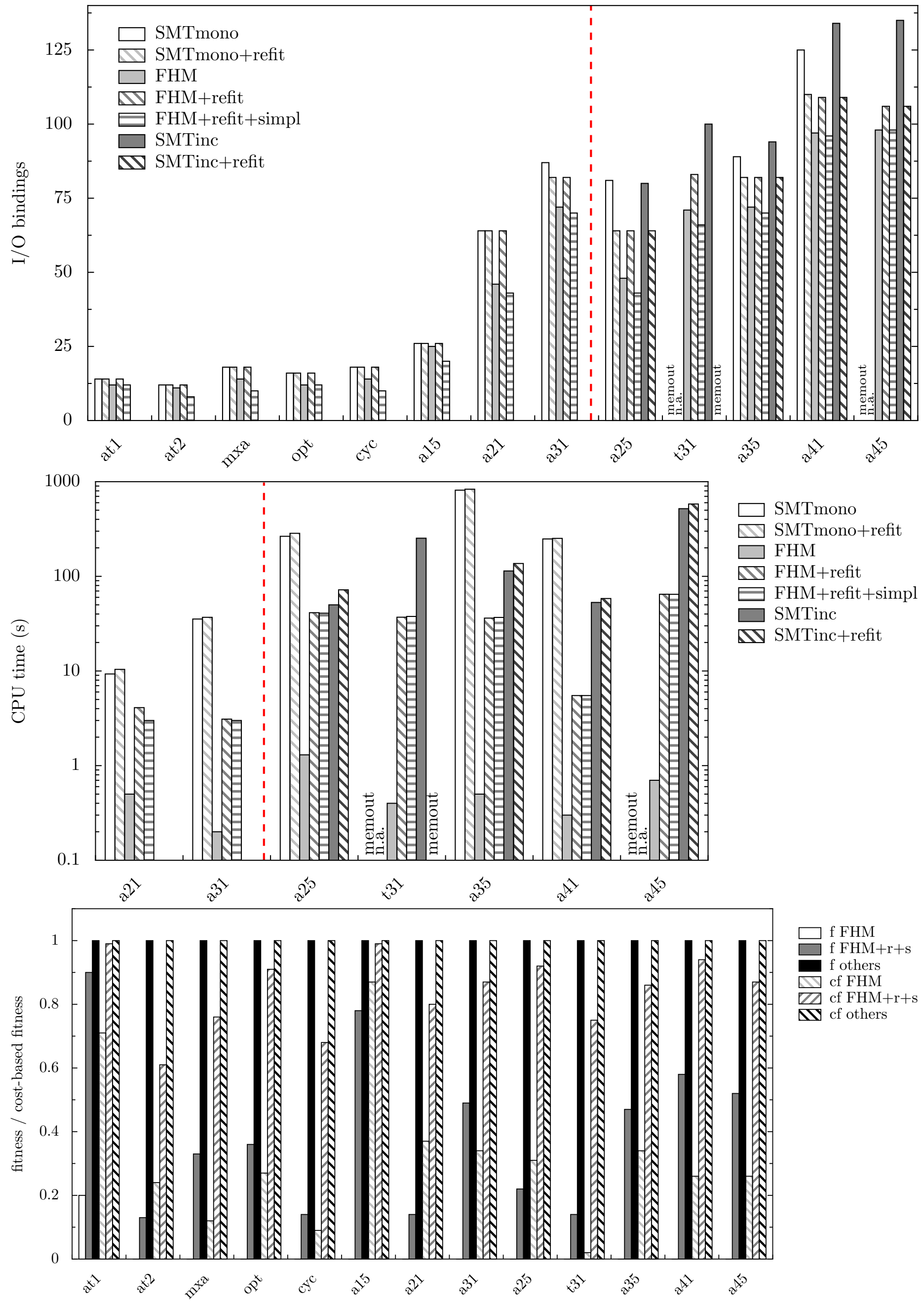

Fig. 5. Comparison between discovery methods.

of the algorithm can be evaluated by comparing the results of SMTmono and SMTinc with their *+refit* versions. These results can be found on Table 2, where *arc* is the number of arcs, $|IO|$ are the number of input/output bindings, T_r is the time to complete of the refit algorithm, T is the total time of each method and *cl* is the fraction of clusters successfully processed by SMTinc.

Since SMT-based methods derive fitting models with minimal number of arcs, only a minimization of the bindings is performed by the refit algorithm. On this regard we can see that SMTmono performs quite a good job for the smallest benchmarks (on the left of the graph, where the refit algorithm could not suppress any binding), but tends to create redundant bindings for the largest ones, specially when there are several iterations of a loop in which there is a choice or constructions like the one in Fig. 4. For these benchmarks there is a variable reduction of bindings ranging from a modest 6% (a31) to a significant 21% (a25). Note that in some of the largest benchmarks (t31 and a45) SMTmono is not able to complete due to memory exhaustion.

The incremental approach SMTinc can handle larger logs, but the C-nets generated are sub-optimal in the number of arcs and bindings they contain [8]. Since this technique is not interesting for small logs, only the largest benchmarks have been used with SMTinc. To distinguish the set of benchmarks in which SMTinc has been executed we have drawn a vertical dashed line to separate both sets. As expected, the number of bindings provided by SMTinc is generally larger than the ones found by SMTmono (when SMTmono is able to complete). However the refit algorithm is capable of minimizing the number of bindings to the same number as SMTmono+refit. In fact, all the C-nets obtained by SMTmono+refit and SMTinc+refit are isomorphic with the only exception of a41, where only 91% of the bindings are shared between both C-nets. The fraction of shared bindings between the C-nets of the *+refit* versions of the discovery methods can be found in Table 4. Formally, the values in the table correspond to the *input/output binding similarity (iobs)* metric defined in [8].

In terms of running time (Fig. 5(center)), the overhead of the refit algorithm is small when compared with the time taken by the discovery algorithm, being usually an order of magnitude smaller (note that the Y axis in the graph is logarithmic). The benchmarks that are not show in the graph had running times of 0.2 s and below.

There are a couple of effects that deserve further comments. For instance for the a41 benchmark, which is the only one in which SMTmono+refit and SMTinc+refit yield different C-nets, the final number of bindings for SMTinc+refit is 109, while in SMTmono+refit the same benchmark has 110 bindings. The reason for this effect is simple: the set of initial bindings in both cases (in the C-nets produced by SMTmono and SMTinc) is different, and since our technique simply seeks for a minimal subset, this minimal subset can be different if the initial sets are also different.

The second particularity is the t31 benchmark. This log is specially difficult for SMT-based techniques, due to the large sequences it contains. Trying to discover a C-net using SMTmono, exhausts the memory, while SMTinc is

	SMTmono			SMTmono+refit						SMTinc				SMTinc+refit			
Log	arc	$\lvert IO\rvert$	T	arc	$\lvert IO\rvert$	T_r	T	f	cf	cl	arc	$\lvert IO\rvert$	T	arc	$\lvert IO\rvert$	T_r	T
at1	6	14	0.0	6	14	0.0	0.0	1.00	1.00								
at2	6	12	0.1	6	12	0.1	0.2	1.00	1.00								
mxa	8	18	0.0	8	18	0.2	0.2	1.00	1.00								
opt	9	16	0.0	9	16	0.1	0.1	1.00	1.00								
cyc	9	18	0.1	9	18	0.1	0.2	1.00	1.00								
a15	14	26	0.0	14	26	0.1	0.1	1.00	1.00								
a21	34	64	9.3	34	64	1.1	10.4	1.00	1.00								
a31	46	87	35.4	46	**82**	1.5	36.9	1.00	1.00								
t31	–	memout		–	–	–	–	0.93	0.97	**2/3**	45	100	253	memout			
a41	63	125	248.4	63	**110**	3.5	251.9	1.00	1.00	3/3	65	134	53	**63**	**109**	5.3	58
a25	34	81	264.9	34	**64**	20.0	284.9	1.00	1.00	8/8	34	80	50	34	**64**	22.0	72
a35	46	89	814.0	46	**82**	19.2	833.2	1.00	1.00	8/8	46	94	114	46	**82**	22.7	137
a45	–	memout		–	–	–	–	1.00	1.00	8/8	65	135	519	**62**	**106**	62.2	581

Table 2. Minimization of the input/output bindings of the C-nets generated using a monolithic SMT-based approach [7], and an incremental one [8].

able to process two of the three clusters in which it is partitioned and fails in the one that contains the larger sequences (thus the C-net found by SMTinc in this benchmark has $f = 0.93$ and $cf = 0.97$). The refit algorithm finds that not all sequences can be replayed by the initial C-net, and the SMT-problem using the C-net as initial skeleton in unfeasible. When it tries to introduce the missing sequences back into the model, simulation proceeds normally and the seven non-replayable sequences are detected, the immediately follows C-net for them is built, united with the original C-net and then the resulting net is minimized in the number of arcs. However, this last phase cannot be completed in the available resources since the immediately follows C-net introduces too many arcs making the SMT problem to be solved of a similar complexity to the one that SMTmono had to solve.

FHM: The refit algorithm is fully exercised when the C-nets found by the FHM plugin are used. This algorithm produces C-nets in a fraction of the times required by SMTmono or SMTinc, but, on the other hand, does not guarantee a fitting model (see Fig. 5(bottom), where in several cases $f = 0.0$). When the refit algorithm is used in these C-nets, the produced models obtain a fitness of 1.0, as expected, at the cost of having a larger (and different) set of bindings (see Fig. 5(top)). The time required by the algorithm is quite reasonable (although much larger than the time spent by FHM), specially when we compare the outcome of the combination FHM+refit in contrast with SMTmono+refit or SMTinc+refit, improving the total running time in one or two orders of magnitude. Thus, it is clear that a hybrid method FHM+refit would be an interesting technique to obtain high quality C-nets with moderate running times. The detailed results can be found on Table 3.

	FHM					FHM+refit						FHM+refit+simpl				
Log	arc	$\lvert IO\rvert$	T	f	cf	arc	$\lvert IO\rvert$	T_r	T	f	cf	arc	$\lvert IO\rvert$	T_{r+s}	f	cf
at1	6	12	0.1	0.20	0.71	6	14	0.6	0.7	**1.0**	1.0	6	12	0.1	0.90	0.99
at2	7	11	0.0	0.00	0.24	6	12	0.2	0.2	**1.0**	1.0	4	8	0.2	0.13	0.61
mxa	8	14	0.0	0.00	0.12	8	18	0.1	0.1	**1.0**	1.0	6	10	0.1	0.33	0.76
opt	7	12	0.0	0.00	0.27	9	16	0.1	0.1	**1.0**	1.0	7	12	0.1	0.36	0.91
cyc	9	14	0.1	0.00	0.09	9	18	0.1	0.2	**1.0**	1.0	5	10	0.1	0.14	0.68
a15	14	25	0.2	0.00	0.87	14	26	0.1	0.3	**1.0**	1.0	11	20	0.1	0.78	0.99
a21	34	46	0.5	0.00	0.37	34	64	3.6	4.1	**1.0**	1.0	25	43	3.0	0.14	0.80
a31	46	72	0.2	0.00	0.34	46	82	2.9	3.1	**1.0**	1.0	40	70	3.0	0.49	0.87
t31	45	71	0.4	0.00	0.02	45	83	36.6	37.0	**1.0**	1.0	36	66	37.7	0.14	0.75
a41	62	97	0.3	0.00	0.26	63	109	5.2	5.5	**1.0**	1.0	57	96	5.5	0.58	0.94
a25	34	48	1.3	0.00	0.31	34	64	40.0	41.3	**1.0**	1.0	25	43	40.8	0.22	0.92
a35	46	72	0.5	0.00	0.34	46	82	35.7	36.2	**1.0**	1.0	40	70	36.8	0.47	0.86
a45	62	98	0.7	0.00	0.26	62	106	64.0	64.7	**1.0**	1.0	58	98	64.6	0.52	0.87

Table 3. Refitting of the C-nets generated using the flexible heuristics miner [5].

To exercise the refit algorithm on the part described in Sect. 3.5 we have executed again the refit algorithm on the C-nets generated by FHM, providing an f parameter to the algorithm that produces C-nets with at most the same amount of bindings than the original FHM-generated nets. We have labeled these C-nets as *FHM+refit+simpl.* The algorithm always achieves the same or less number of bindings than the original FHM in a processing time that is almost the same as the refit algorithm without simplification[5]. However, the fitness results are much better than the ones obtained by FHM, on the number of replayable sequences as well as on the cost-based fitness.

5 Conclusion and Future Work

This paper describes an SMT-based enhancement algorithm to improve the quality of a given C-net with respect to a log. The technique has been implemented and tested with the existing C-net discovery algorithms in the literature. The experimental results have shown that the combination of fast and heuristically based methods with SMT-based techniques can yield high quality C-nets much faster than purely SMT-based methods. As future work, we plan to study to which extent the simplification procedure can produce more valuable models when the logs contain noise.

Acknowledgements

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

[5] In some benchmarks FHM+refit+simpl has less execution time than the refit algorithm without simplification, this is due to the small variability of the SMT solver.

Log	FHM vs. SMTmono	FHM vs. SMTinc	SMTmono vs. SMTinc
at1	1.00	–	–
at2	1.00	–	–
mxa	1.00	–	–
opt	1.00	–	–
cyc	1.00	–	–
a15	1.00	–	–
a21	1.00	–	–
a25	1.00	1.00	1.00
a31	1.00	–	–
a35	1.00	1.00	1.00
t31	–	–	–
a41	1.00	0.91	0.91
a45	–	1.00	–

Table 4. Similarity between the obtained C-nets, after having applied the refit algorithm.

References

1. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. van der Aalst, W., Adriansyah, A., van Dongen, B.: Causal nets: a modeling language tailored towards process discovery. In: CONCUR. (2011) 28–42
3. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Genetic process mining. In: ICATPN. Volume 3536 of LNCS. (2005) 48–69
4. Weijters, A., van der Aalst, W., de Medeiros, A.A.: Process mining with the heuristics miner-algorithm. Technical Report WP 166, BETA Working Paper Series, Eindhoven University of Technology (2006)
5. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: CIDM, IEEE (2011) 310–317
6. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Information and Systems **33**(1) (2008) 64–95
7. Solé, M., Carmona, J.: An SMT-based discovery algorithm for C-nets. In: ATPN'12 (accepted for publication), available as technical report LSI-12-2-R at http://www.lsi.upc.edu/dept/techreps/techreps.html (2012)
8. Solé, M., Carmona, J.: A high-level strategy for C-net discovery. In: ACSD'12 (accepted for publication). (2012)
9. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE TKDE **16**(9) (2004) 1128–1142
10. Jha, S., Limaye, R., Seshia, S.: Beaver: Engineering an efficient SMT solver for bit-vector arithmetic. In: Computer Aided Verification. (2009) 668–674
11. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: ATPN. (2008) 368–387
12. Adriansyah, A., van Dongen, B., van der Aalst, W.: Conformance checking using cost-based fitness analysis. In: EDOC. (2011) 55 –64