Directed Unfolding of Petri Nets

Blai Bonet[1], Patrik Haslum[2], Sarah Hickmott[3], and Sylvie Thiébaux[2]

[1] Universidad Simón Bolívar, Departamento de Computación, Caracas, Venezuela,
[2] National ICT Australia & The Australian National University, Canberra, Australia
[3] National ICT Australia & The University of Adelaide, Adelaide, Australia

Abstract. The key to efficient on-the-fly reachability analysis based on unfolding is to focus the expansion of the finite prefix towards the desired marking. However, current unfolding strategies typically equate to blind (breadth-first) search. They do not exploit the knowledge of the marking that is sought, merely entertaining the hope that the road to it will be short. This paper investigates *directed unfolding*, which exploits problem-specific information in the form of a heuristic function to guide the unfolding towards the desired marking. In the unfolding context, heuristic values are estimates of the distance between configurations. We show that suitable heuristics can be automatically extracted from the original net. We prove that unfolding can rely on heuristic search strategies while preserving the finiteness and completeness of the generated prefix, and in some cases, the optimality of the firing sequence produced. We also establish that the size of the prefix obtained with a useful class of heuristics is never worse than that obtained by blind unfolding. Experimental results demonstrate that directed unfolding scales up to problems that were previously out of reach of the unfolding technique.

1 Introduction

The Petri net unfolding process, originally introduced by McMillan [1], has gained the interest of researchers in verification (see e.g. [2]), diagnosis [3] and, more recently, planning [4]. All have reasons to analyse reachability in distributed transition systems, looking to unfolding for some relief of the state explosion problem. Unfolding a Petri net reveals all possible partially ordered runs of the net, without the combinatorial interleaving of independent events. Whilst the unfolding can be infinite, McMillan identified the possibility of a finite prefix with all reachable states. Esparza, Römer and Vogler generalised his approach, to produce the now commonly used ERV unfolding algorithm [5]. This algorithm involves a search, but does not mandate a specific search strategy. Typically, it has been implemented as a breadth-first search, using the length of paths to select the next node to add and to determine cut-off events.

Of the various unfolding-based reachability techniques, experimental results indicate on-the-fly analysis to be most efficient for proving the reachability of a single marking [6]. Nevertheless, generating the complete prefix up to a particular state via breadth-first search quickly becomes impractical when the unfolding is wide or the shortest path to the state is deep. Unfortunately, it has not been

obvious what other strategies could be used in the ERV algorithm and recent results have shown that the use of depth-first search in a simpler unfolding algorithm is incorrect [7]. In this paper, we investigate *directed unfolding*, a strategy that takes advantage of information about the sought marking to guide the search. The reason why such an informed strategy has not been considered before may be that unfolding is typically used to prove the absence of deadlocks: this has set the focus on making the entire prefix smaller rather than on reducing the part of the search space explored to reach a particular marking. However, as demonstrated below, information about the goal marking can help also in the case when this marking is not reachable.

Inspired by heuristic search in artificial intelligence, particularly in the area of automated planning, directed unfolding exploits problem-specific information in the form of a heuristic function to guide search towards the desired marking. Specifically, the heuristic estimates the shortest distance from a given marking to the desired one, and is used to implement a search strategy where choices are explored in increasing order of their estimated distance. If the heuristic is sufficiently informative, this order provides effective guidance towards the marking sought. Whilst the order is not always adequate, in the sense defined in [5], it still guarantees finiteness and completeness of the generated prefix. Interestingly, our proof relies on the observation that adequate orders are stronger than necessary for these purposes, and introduces the weaker notion of semi-adequate ordering.

Using heuristics, automatically extracted from the representation of a transition system, to guide search has significantly improved the scalability of automated planning [8–10]. We show that heuristic values can be similarly calculated from a Petri net. If the chosen heuristic is *admissible* (meaning it never overestimates the shortest distances) then directed unfolding finds the *shortest* path to the target marking, just like breadth-first search. Moreover, a slightly stronger property than admissibility guarantees that the prefix produced is never larger than the prefix obtained by breadth-first search. Using inadmissible heuristics, completeness and correctness are preserved, and performance is often dramatically improved at the expense of optimality. Altogether, directed unfolding can solve much larger problems than the original breadth-first ERV algorithm. Moreover, its implementation requires only minor additions.

The paper is organised as follows. Section 2 is an overview of Place/Transition nets, unfoldings, and on-the-fly reachability analysis. Section 3 describes the ideas behind directed unfolding and establishes its theoretical properties. In Section 4, we show how to automatically extract a range of heuristics from the Petri net description. In Section 5 presents experimental results and Section 6 concludes with remarks about related and future work.

2 Petri Nets, Unfolding and Reachability Analysis

2.1 Place/Transition Petri Nets

Petri nets provide a factored representation of discrete-event systems. States are not enumerated and flattened into single unstructured entities but rather

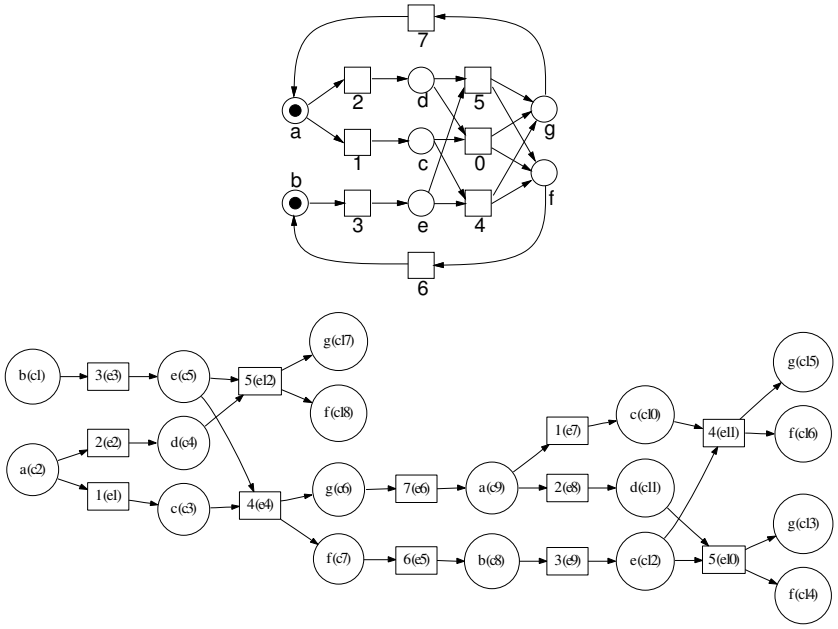

Fig. 1. Example of a Place/Transition Net (top) and its unfolding (bottom).

explicitly factorized into variables (places) such that the temporal relations between variables become transitions that produce and consume markers in the net. We consider the so-called *Place/Transition* (P/T) nets, and describe them only briefly; a detailed exposition can be found in [11].

A P/T-net (top part of Figure 1) consists of a net N and its initial marking M_0. The net is a directed bipartite graph where the nodes are places and transitions (depicted as circles and squares respectively). Typically, places represent the state variables and transitions the events of the underlying discrete-event system. The dynamic behaviour is captured by the flow relation F between places and transitions and vice versa. The *marking* of a P/T-net represents the state of the system. It assigns to each place zero or more tokens (depicted as dots).

Definition 1. *A P/T-net is a 4-tuple* (P, T, F, M_0) *where* P *and* T *are disjoint finite sets of places and transitions, respectively,* $F : (P \times T) \cup (T \times P) \to \{0, 1\}$ *is a flow relation indicating the presence (1) or absence (0) of arcs, and* $M_0 : P \to \mathbb{N}$ *is the initial marking.*

The *preset* ${}^\bullet x$ of node x is the set $\{y \in P \cup T : F(y, x) = 1\}$, and its *postset* $x^\bullet$ is the set $\{y \in P \cup T : F(x, y) = 1\}$. The marking M enables a transition t if $M(p) > 0$ for all $p \in {}^\bullet t$. The occurrence, or *firing*, of an enabled transition t absorbs a token from each of its preset places and puts one token in each postset place. This corresponds to a state transition in the modeled system, moving the

net from M to the new marking M' given by $M'(p) = M(p) - F(p,t) + F(t,p)$ for each p; this is denoted as $M \xrightarrow{t} M'$. A *firing sequence* $\sigma = t_1 \dots t_n$ is a legal sequence of transition firings, i.e. there are markings $M_1, \dots, M_n$ such that $M_0 \xrightarrow{t_1} M_1 \cdots M_{n-1} \xrightarrow{t_n} M_n$; this is denoted as $M_0 \xrightarrow{\sigma} M_n$. A marking M is *reachable* if there exists a firing sequence σ such that $M_0 \xrightarrow{\sigma} M$. In this paper we only consider 1-bounded nets, meaning that all reachable markings assign at most one token at each place.

2.2 Unfolding

Unfolding is a method for reachability analysis which exploits and preserves concurrency information in the Petri net. It a partially ordered structure of events that represents all possible firing sequences of the net from the initial marking.

Unfolding a P/T-net produces a pair $U = (ON, \varphi)$ where $ON = (B, E, F')$ is an occurrence net, which is a P/T-net without cycles, self conflicts or backward conflicts (defined below), and φ is a homomorphism from ON to N that associates the places/transitions of ON with the places/transitions of the P/T-net.

A node x is in self conflict if there exist two paths to x which start at the same place and immediately diverge. A backward conflict happens when two transitions output to the same place. Such cases are undesirable since in order to decide whether a token can reach a place in backward conflict, it would be necessary to reason with disjunctions such as from which transition the token came. Therefore, the process of unfolding involves breaking all backward conflicts by making independent copies of the places involved in the conflicts, and thus the occurrence net ON may contain multiples copies of places and transitions of the original net which are identified with the homomorphism.

In the occurrence net ON, places and transitions are called conditions B and events E respectively. The initial marking M_0 defines a set of initial conditions B_0 in ON such that the places initially marked are in 1-1 correspondence with the conditions in B_0. The set B_0 constitutes the "seed" of the unfolding.

The bottom part in Figure 1 shows a prefix of the unfolding of the P/T-net in the top part. Note the multiple instances of place g, for example, due to the different firing sequences through which it can be reached (multiple backward conflicts). Note also that transition 0 does not appear in the unfolding, as there no firing sequence that enables transition 0.

2.3 Configurations

To understand how a prefix of an unfolding is built, the most important notions are that of a configuration and local configuration. A *configuration* represents a possible partially ordered run of the net. It is a finite set of events C such that:

1. C is causally closed: $e \in C \Rightarrow e' \in C$ for all $e' \leq e$,
2. C contains no forward conflict: ${}^\bullet e_1 \cap {}^\bullet e_2 = \emptyset$ for all $e_1 \neq e_2$ in C;

where $e' \leq e$ means there is a directed path from e' to e in ON. If these two conditions are met, the events in a configuration C can be ordered into a firing sequence with respect to B_0. For instance, in the finite prefix in Figure 1, $\{\mathsf{e1}, \mathsf{e3}, \mathsf{e4}\}$ is a configuration, while $\{\mathsf{e1}, \mathsf{e4}\}$ and $\{\mathsf{e1}, \mathsf{e2}\}$ are not since the former is not causally closed and the latter has a forward conflict.

A configuration C can be associated with a final marking $\text{Mark}(C)$ of the original P/T-net by identifying which conditions will contain a token after the events in C are fired from the initial conditions; i.e. $\text{Mark}(C) = \varphi((B_0 \cup C^\bullet) \setminus {}^\bullet C)$ where $C^\bullet$ (resp. ${}^\bullet C$) is the union of postsets (resp. presets) of all events in C. In other words, the marking of C identifies the resultant marking of the original P/T-net when only the transitions labelled by the events in C occur. For instance, in Figure 1, the marking of configuration $\{\mathsf{e1}, \mathsf{e3}, \mathsf{e4}, \mathsf{e5}\}$ is $\{\mathsf{g}, \mathsf{b}\}$. The *local configuration* of an event e, denoted by $[e]$, is the minimal configuration containing event e. For example, $[\mathsf{e5}] = \{\mathsf{e1}, \mathsf{e3}, \mathsf{e4}, \mathsf{e5}\}$. A set of events can occur in the same firing sequence iff the union of their local configurations is a configuration.

2.4 Finite Complete Prefix

The unfolding process involves identifying which transitions are enabled by those conditions, currently in the occurrence net, that can be simultaneously marked. These are referred to as the possible next events. A new instance of each is added to the occurrence net, as are instances of the places in their postsets.

The unfolding process starts from the seed B_0 and extends it iteratively. In most cases, the unfolding U is infinite and thus cannot be built. However, it is not necessary to build U entirely, but only a *complete finite prefix* β of U that contains all the information in U. Formally, a prefix β of U is *complete* if for every reachable marking M, there exists a configuration $C \in \beta$ such that $\text{Mark}(C) = M$, and for every transition t enabled by M there is an event $e \notin C$ with $\varphi(e) = t$ such that $C \cup \{e\}$ is a configuration.

The key for obtaining a complete finite prefix is to identify those events at which the unfolding can be ceased without loss of information. Such events are referred to as *cut-off events* and can be defined in terms of an *adequate order* on configurations [1, 5, 12]. In the following, $C \oplus E$ denotes a configuration that extends C with the finite set of events E disjoint from C; such E is called an extension of configuration C.

Definition 2 (Adequate Orderings). *A strict partial order $\prec$ on finite configurations is an* adequate order *if and only if*

(*a*) *$\prec$ is well founded, i.e. it has no infinite descending chains,*
(*b*) *$C_1 \subset C_2 \Rightarrow C_1 \prec C_2$, and*
(*c*) *$\prec$ is* weakly *preserved by finite extensions; i.e. if $C_1 \prec C_2$ and $Mark(C_1) = Mark(C_2)$, then for all finite extension E_2 of C_2, there exist a finite extension E_1 of C_1 that is structurally isomorphic[1] to E_2, and $C_1 \oplus E_1 \prec C_2 \oplus E_2$.*

[1] Two extensions E and E' are structurally isomorphic if the labelled digraphs induced by the two sets of events and their adjacent conditions are isomorphic [12].

Algorithm 1 The ERV Unfolding Algorithm (and ERV/fly variant)

Input: a P/T-net (P, T, F, M_0) (and transition t_R for ERV/fly).
Output of ERV: complete finite prefix β.
Output of ERV/fly: finite prefix β with event e_R, with $\varphi(e_R) = t_R$, if t_R is reachable,
finite prefix β with no event e_R, with $\varphi(e_R) = t_R$, otherwise.

1. Initialise the prefix β with the conditions in B_0
2. Initialise the priority queue with the events possible in B_0
3. Initialise the set *cut-off* to $\emptyset$
4. **while** the queue is not empty **do**
5. Remove event e in the queue (minimal with respect to $\prec$)
6. ⟦only for ERV/fly⟧ **if** $h([e]) = \infty$ **then terminate** (t_R is not reachable)
7. **if** $[e]$ contains no event in *cut-off* **then**
8. Add e and conditions for its postset to β
9. ⟦only for ERV/fly⟧ **if** $\varphi(e) = t_R$ **then terminate** (t_R is reachable)
10. Identify the *new* possible next events and insert them in the queue
11. **if** e is a cut-off event in β with respect to $\prec$ **then**
12. Update *cut-off* $:=$ *cut-off* $\cup \{e\}$
13. **endif**
14. **endif**
15. **endwhile**

Without threat to completeness, we can cease unfolding from an event e, if it takes the net to a marking which can be caused by some other already unfolded event e' such that $[e'] \prec [e]$. This is because the events (and thus marking) which proceed from e will also proceed from e'. Relevant proofs can be found in [5, 12].

Definition 3 (Cut-off Events). *Let $\prec$ be an adequate order and β a prefix. An event e is a* cut-off event in β with respect to $\prec$ *iff β contains some event e' such that $Mark([e]) = Mark([e'])$ and $[e'] \prec [e]$.*

2.5 The ERV Algorithm

Algorithm 1 shows the well-known ERV algorithm for unfolding P/T-nets [5] (and a variant, called ERV/fly, which will be discussed later). ERV maintains a queue of events, sorted in increasing order with respect to $\prec$. At each iteration, a minimal event in the queue is processed, starting with checking whether its local configuration contains any cut-off event with respect to $\prec$ in the prefix β under construction. If not, the event is added to the prefix along with conditions for its postset, and the new possible next events enabled by the new conditions are inserted in the queue. The algorithm terminates when all queue events have been processed (the ERV/fly variant has two additional conditions for earlier termination). This is the ERV algorithm exactly as it is described in [5].

It is important to mention certain details about the implementation of the algorithm. First, the order $\prec$ is used *both to order the queue and to identify the cut-off events*. As noted in [5], this implies that if the ordering $\prec$ is total, the

check at line 11 in Algorithm 1 ("e is a cut-off event in β with respect to $\prec$") can be replaced by the simpler check: "β contains a local configuration $[e']$ such that $\text{Mark}([e]) = \text{Mark}([e'])$", since with a total order, $[e'] \prec [e]$ for any event e that is dequeued after e'. This optimisation may be important if evaluating $\prec$ is expensive (this, however, is not the case for any order we consider in this paper). Second, it is in fact not necessary to insert new events that are causal successors of a cut-off event into the queue – which is done in the algorithm as described – since they will only be discarded when dequeued. While this optimisation makes no difference to the prefix generated, it may have a significant impact on both runtime and memory use. For optimizations related to the generation of possible next events see [13].

Besides the explicit input parameters, the ERV and ERV/fly algorithms implicitly depend on an order $\prec$ (and also on a function h for ERV/fly). Whenever this dependency needs to be emphasized, we will refer to both algorithms as ERV$[\prec]$ and ERV/fly$[\prec, h]$ respectively. Again, note that whatever order this may be, *it is used both to order the queue and to identify cut-off events.*

MOLE[2] is a freeware program that implements the ERV algorithm for 1-bounded P/T-nets. MOLE uses McMillan's cardinality-based ordering ($C \prec_m C'$ iff $|C| < |C'|$) [1], further refined into a total order [5]. Note that using this order equates to a breadth-first search strategy. MOLE implements the optimisation described above, i.e. successors of cut-off events are never placed on the queue.

The prefix in Figure 1 is the complete finite prefix that MOLE generates for our example. The events e10, e11, and e12 are all cut-off events. This is because each of their local configurations has the same marking as the local configuration of event e4, i.e. $\{f, g\}$, and each of them is greater than the local configuration of e4 with respect to the adequate order implemented by MOLE.

2.6 On-The-Fly Reachability Analysis

We define the *reachability problem* (also often called coverability problem) for 1-bounded P/T-nets as follows:

> REACHABILITY: Given a P/T-net (P, T, F, M_0) and a subset $P' \subseteq P$, determine whether there is a firing sequence σ such that $M_0 \xrightarrow{\sigma} M$ where $M(p) = 1$ for all $p \in P'$.

This problem is PSPACE-complete [14].

Since unfolding constructs a complete finite prefix that represents every reachable marking by a configuration, it can be used as the basis of an algorithm for deciding REACHABILITY. However, deciding if the prefix contains any configuration that leads to a given marking is still NP-complete [6]. If we are interested in solving multiple REACHABILITY problems for the same net and initial marking, this is still an improvement. Algorithms taking this approach have been designed using mixed-integer linear programming [15], stable models for Logic Programs [16], and other methods [6, 17].

[2] http://www.fmi.uni-stuttgart.de/szs/tools/mole/

However, if we are interested in the reachability of just one single marking, the form of completeness offered by the prefix constructed by unfolding is unnecessarily strong: we require only that the target marking is represented by some configuration if it is indeed reachable. We will refer to this weaker condition as *completeness with respect to the goal marking*. This was recognised already by McMillan, who suggested an on-the-fly approach to reachability. It involves introducing a new transition t_R to the original net with ${}^\bullet t_R = P'$ and $t_R{}^\bullet = \{p_R\}$ where p_R is a *new place*. [3] The net is then unfolded until an event e_R, such that $\varphi(e_R) = t_R$, is retrieved from the queue. At this point we can conclude that the set of places P' is reachable. If unfolding terminates without identifying such an event, P' is not reachable. If $[e_R]$ is not required to be the shortest possible firing sequence, it is sufficient to stop as soon as e_R is generated as one of the possible next events, but to guarantee optimality, even with breadth-first unfolding, it is imperative to wait until the event is pulled out of the queue. Experimental results have shown the on-the-fly approach to be most efficient for deciding the reachability of a single marking [6].

The ERV/fly variant of the ERV unfolding algorithm embodies two "short cuts", in the form of conditions for earlier termination, which are motivated by the fact that we are interested only in completeness with respect to the goal marking. The first is simply to adopt McMillan's on-the-fly approach, stopping when an instance of transition t_R is dequeued. The second depends on a property of the heuristic function h, and will be discussed in Section 3.3. [4]

3 Directing the Unfolding

In the context of the reachability problem, we are only interested in checking whether the transition t_R is reachable. An unfolding algorithm that doesn't use this information is probably not the best approach. In this section, we aim to define a *principled method* for using this information during the unfolding process in order to solve the reachability problem more efficiently. The resulting approach is called "directed unfolding" as opposed to the standard "blind unfolding".[5]

The basic idea is that for deciding Reachability, the unfolding process can be understood as a *search process* on the quest for t_R. Thus, when selecting events from the queue, we should favor those "closer" to t_R as their systematic exploration results in a more efficient search strategy. This approach is only

[3] Strictly speaking, to preserve 1-safeness, it is also necessary to add a new place complementary to p_R to ${}^\bullet t_R$ to avoid multiple firings of t_R.

[4] In addition, for completeness with respect to a single goal marking, it is not necessary to insert cut-off events into the prefix at all, since any marking represented by the local configuration of a cut-off event is by definition already represented by another event. This optimisation may not have a great impact on runtime, at least if the previously described optimisation of not generating successors of cut-off events is already in place, but may reduce memory requirements.

[5] The term "directed" has been used elsewhere to emphasize the informed nature of other model-checking algorithms [18].

possible if the prefix constructed is guaranteed to be complete, in the sense that it will, eventually, contain an instance of t_R if it is reachable.

We show that the ERV algorithm can be used with the same definition of cut-off events when the notion of adequate orderings is replaced by a weaker notion that we call *semi-adequate* orderings. This is prompted by the observation that the definition of adequate orderings is a sufficient but not a necessary condition for a sound definition of cut-off events. Indeed, just replacing condition (b) in Definition 2 by a weaker condition opens the door for a family of semi-adequate orderings that allow us to direct the unfolding process.

3.1 Principles

As is standard in state-based search, our orderings are constructed upon the values of a function f that maps configurations into non-negative numbers (including infinity). Such functions f are composed of two parts $f(C) = g(C)+h(C)$ in which $g(C)$ refers to the "cost" of C and $h(C)$ estimates the distance from $\mathrm{Mark}(C)$ to the *target marking* $\{p_R\}$. For the purposes of the present work, we will assume a fixed function $g(C) = |C|$, yet other possibilities also make sense, e.g. when transitions are associated with costs, and the cost of a set of transitions is defined as the sum of the costs of the transitions in the set.

The function $h(C)$ is a non-negative valued function on configurations, and is required to satisfy:

1. $h(C) = 0$ if $\mathrm{Mark}(C)$ contains a condition c_R such that $\varphi(c_R) = p_R$ where p_R is the new place in ${t_R}^\bullet$, and
2. $h(C) = h(C')$ whenever $\mathrm{Mark}(C) = \mathrm{Mark}(C')$.

Such functions will be called *heuristic functions on configurations*. Note that the function which assigns value 0 to all configurations is a heuristic function. We will denote this function $h \equiv 0$. For two heuristic functions, $h \leq h'$ denotes the standard notion of $h(C) \leq h'(C)$ for all configurations C.

Let h be a heuristic and, for $f(C) = |C| + h(C)$, define the ordering $\prec_h$ as follows:

$$C \prec_h C' \quad \text{iff} \quad \begin{cases} f(C) < f(C') & \text{if } f(C) \neq f(C') \\ |C| < |C'| & \text{if } f(C) = f(C'). \end{cases}$$

Observe that $\prec_{h\equiv 0}$ is the strict partial order $\prec_m$ on configurations used by McMillan [1], which can be refined into the total order defined in [5].

Let us define $h^*(C) = |C'|-|C|$, where $C' \supseteq C$ is a configuration of minimum cardinality that contains an instance of t_R if one exists, and ∞ otherwise. (By "an instance of t_R" we mean of course an event e_R such that $\varphi(e_R) = t_R$.) We then say that h is an *admissible heuristic* if $h(C) \leq h^*(C)$ for all finite configurations C. Likewise, let us say that a finite configuration C^* is *optimal* if it contains an instance of t_R, and it is of minimum cardinality among such configurations. By f^* we denote $|C^*|$ if an optimal configuration exists (i.e. if t_R is reachable) and ∞ otherwise. In the following, $\mathsf{ERV}[h]$ denotes $\mathsf{ERV}[\prec_h]$.

Theorem 1 (Main). *Let h be a heuristic function on configurations. Then,* ERV$[h]$ *computes a finite and complete prefix of the unfolding. Furthermore, if h is admissible, then* ERV$[h]$ *finds an optimal configuration if t_R is reachable. Both claims also hold for any semi-adequate ordering that refines $\prec_h$.*[6]

Note that this result by no means contradicts a recent proof that unfolding with depth-first search is incorrect [7]: Not only do heuristic strategies have a "breadth" element to them which depth-first search lacks, but, more importantly, the algorithm shown incorrect differs from the ERV algorithm in that when identifying cut-off events it only checks if the prefix contains a local configuration with identical marking but does not check whether the ordering $\prec$ holds.

Optimal configurations are important in the context of diagnosis since they provide shortest firing sequences to reach a given marking, e.g. a faulty state in the system. A consequence of Theorem 1 is that the MOLE implementation of the ERV algorithm, which equates using a refinement of $\prec_{h\equiv 0}$ into a total order [5], finds shortest firing sequences. In the next two sections, we will give examples of heuristic functions, both admissible and non-admissible, and experimental results on benchmark problems. In the rest of this section, we provide the technical characterization of semi-adequate orderings and their relation to adequate ones, as well as the proofs required for the main theorem. We also provide a result concerning the size of the prefixes obtained.

3.2 Technical Details

Upon revising the role of adequate orders when building the complete finite prefix, we found that condition (b), i.e. $C \subset C' \Rightarrow C \prec C'$, in Definition 2 is only needed to guarantee the finiteness of the generated prefix. Indeed, let n be the number of reachable markings and consider an infinite sequence of events $e_1 < e_2 < \cdots$ in the unfolding. Then, there are $i < j \leq n+1$ such that $\text{Mark}([e_i]) = \text{Mark}([e_j])$, and since $[e_i] \subset [e_j]$, condition (b) implies $[e_i] \prec [e_j]$ making $[e_j]$ into a cut-off event, and thus the prefix is finite [5]. A similar result can be achieved if condition (b) is replaced by the weaker condition that in every infinite chain $e_1 < e_2 < \cdots$ of events there are $i < j$ such that $[e_i] \prec [e_j]$. To slightly simplify the proofs, we can further weaken that condition by asking that the local configurations of these events have equal markings.

Definition 4 (Semi-Adequate Orderings). *A strict partial order $\prec$ on finite configurations is a* semi-adequate order *if and only if*

(a) *$\prec$ is well founded, i.e. it has no infinite descending chains,*
(b) *in every infinite chain $C_1 \subset C_2 \subset \cdots$ of configurations with equal markings there are $i < j$ such that $C_i \prec C_j$, and*
(c) *$\prec$ is* weakly *preserved by finite extensions.*

Theorem 2 (Finiteness and Completeness). *If $\prec$ is a semi-adequate order, the prefix produced by* ERV$[\prec]$ *is finite and complete.*

[6] Ordering $\prec'$ refines $\prec$ iff $C \prec C'$ implies $C \prec' C'$ for all configurations C and C'.

Proof. The completeness proof is identical to the proof of Proposition 4.9 in [5, p. 14] which states the completeness of the prefix computed by ERV for adequate orderings: this proof does not rely on condition (b) at all. The finiteness proof is similar to the proof of Proposition 4.8 in [5, p. 13] which states the finiteness of the prefix computed by ERV for adequate orderings. If the prefix is not finite, then by the version of König's Lemma for branching processes [19], an infinite chain $e_1 < e_2 < \cdots$ of events exists in the prefix. Each event e_i defines a configuration $[e_i]$ with marking $\mathrm{Mark}([e_i])$, and since the number of markings is finite, there is at least one marking that appears infinitely often in the chain. Let $e'_1 < e'_2 < \cdots$ be an infinite subchain such that $\mathrm{Mark}([e'_1]) = \mathrm{Mark}([e'_j])$ for all $j > 1$. By condition (b) of semi-adequate orderings, there are $i < j$ such that $[e'_i] \prec [e'_j]$ that together with $\mathrm{Mark}([e'_i]) = \mathrm{Mark}([e'_j])$ make e'_j into a cut-off event and thus the chain cannot be infinite. □

Clearly, if $\prec$ is an adequate order, then it is a semi-adequate order. The converse is not necessarily true. The fact that $\prec_h$ is semi-adequate is a consequence of the monotonicity of $g(C) = |C|$, i.e. $C \subset C' \Rightarrow g(C) < g(C')$, and that configurations with equal markings have identical h-values.

Theorem 3 (Semi-Adequacy of $\prec_h$). *If h is a heuristic on configurations, $\prec_h$ is a semi-adequate order.*

Proof. That $\prec_h$ is irreflexive and transitive is direct from definition.

For well-foundedness, first observe that if C and C' are two configurations with the same marking, then $C \prec_h C'$ iff $|C| < |C'|$. Let $C_1 \succ_h C_2 \succ_h \cdots$ be an infinite descending chain of *finite* configurations with markings $M_1, M_2, \ldots$ respectively. Observe that not all C_i's have $f(C_i) = \infty$ since, by definition of $\prec_h$, this would imply $\infty > |C_1| > |C_2| > \cdots \geq 0$ which is impossible. Similarly, at most finitely many C_i's have infinite f-value. Let $C'_1 \succ_h C'_2 \succ_h \cdots$ be the subchain where $f(C'_i) < \infty$ for all i, and $M'_1, M'_2, \ldots$ the corresponding markings. Since the number of markings is finite, we can extract a further subsubchain $C''_1 \succ_h C''_2 \succ_h \cdots$ such that $\mathrm{Mark}(C''_1) = \mathrm{Mark}(C''_j)$ for all $j > 1$. Therefore, $|C''_1| > |C''_2| > \cdots \geq 0$ which is impossible since all C''_i's are finite.

For condition (b), let $C_1 \subset C_2 \subset \cdots$ be an infinite chain of finite configurations with equal markings. Therefore, $val \doteq h(C_1) = h(C_j)$ for all $j > 1$, and also $|C_1| < |C_2|$. If $val = \infty$, then $C_1 \prec_h C_2$. If $val < \infty$, then $f(C_1) = |C_1| + val < |C_2| + val = f(C_2)$ and thus $C_1 \prec_h C_2$.

Finally, if $C_1 \prec_h C_2$ have equal markings and the extensions E_1 and E_2 are isomorphic, the configurations $C'_1 = C_1 \oplus E_1$ and $C'_2 = C_2 \oplus E_2$ also have equal markings, and it is straightforward to show that $C'_1 \prec_h C'_2$. □

Proof (of Theorem 1). That ERV[h] computes a complete and finite prefix is direct since, by Theorem 3, $\prec_h$ is semi-adequate and, by Theorem 2, this is enough to guarantee finiteness and completeness of the prefix.

For the second claim, assume that t_R is reachable. Then, the prefix computed by ERV contains at least one instance of t_R. First, we observe that until e_R is dequeued, the queue always contains an event e such that $[e]$ is a prefix of an

optimal configuration C^*. This property holds at the beginning (initially, the queue contains all possible extensions of the initial conditions) and by induction remains true after each iteration of the while loop. This is because if e is dequeued then either $e = e_R$, or a successor of e will be inserted in the queue which will satisfy the property, or it must be the case that e is identified as a cut-off event by ERV. But the latter case implies that there is some e' in the prefix built so far such that $\text{Mark}([e']) = \text{Mark}([e])$ and $f([e']) < f([e])$. This in turn implies that $h([e']) = h([e])$, and thus $|[e']| < |[e]|$ which contradicts the assumption on the minimality of C^*.

For proof by contradiction, suppose that ERV dequeues a instance e_R of t_R such that $[e_R]$ is not optimal, i.e. not of minimum cardinality. If e is an event in the queue, at the time e_R is dequeued, such that $[e]$ is a subset of an optimal configuration C^*, then

$$f([e]) \;=\; |[e]| + h([e]) \;\leq\; |[e]| + h^*([e]) \;=\; |[e]| + |C^*| - |[e]| \;=\; |C^*|\,.$$

On the other hand, since $[e_R]$ is non-optimal by supposition, $f([e_R]) = |[e_R]| > |C^*|$. Therefore, $f([e_R]) > f([e])$ and thus $[e] \prec_h [e_R]$ and e_R could not have been pulled out of the queue before e.

Observe that the proof does not depend on how the events with equal f-values are ordered in the queue. Thus, any refinement of $\prec_h$ also works. □

3.3 Size of the Finite Prefix

As we have already remarked, to solve Reachability using unfolding we require only that the prefix is complete with respect to the sought marking, i.e. that it contains a configuration representing that marking iff the marking is reachable. This enables us to take certain "short cuts", in the form of conditions for earlier termination, in the unfolding algorithm, which results in a smaller prefix being constructed. In this section, we show first that these modifications preserve the completeness of the algorithm, and the guarantee of finding an optimal solution if the heuristic is admissible. Second, under some additional assumptions, we show a result relating the size of the prefix computed by directed on-the-fly unfolding to the informedness of the heuristic.

Before proceeding, let us review the modifications made in the variant of the ERV algorithm which we call ERV/fly (for ERV on-the-**fly**). The first "short cut" is adopting the on-the-fly approach, terminating the algorithm as soon as an instance of the target transition t_R is added to the prefix. For the second, if the heuristic h has the property that $h(C) = \infty$ implies $h^*(C) = \infty$ (i.e. it is not possible to extend C into a configuration containing an instance t_R), then the unfolding can be stopped as soon as the f-value of the next event retrieved from the queue is ∞, since this implies that t_R is unreachable. We call heuristics that satisfy this property *safely pruning*. Note pruning safety is a weaker requirement than admissibility, in the sense that an admissible heuristic is always safely pruning.

Monotonicity is another well-known property of heuristic functions, which is stronger than admissibility. A heuristic h is *monotonic* iff it satisfies the triangle

inequality $h(C) \leq |C'| - |C| + h(C')$, i.e. $f(C) \leq f(C')$, for all finite $C' \supseteq C$. If h is monotonic, the order $\prec_h$ is in fact adequate [4]. Even though admissibility does not imply monotonicity, it is in practice difficult to construct good admissible heuristics that are not monotonic. The admissible heuristic $h^{\max}$, described in the next section, is also monotonic.

Although ERV/fly depends on an order $\prec$ and a heuristic h, we consider only the case of $\prec_h$ and h for the same heuristic. Thus, we denote with ERV/fly$[h]$ the algorithm ERV/fly$[\prec_h, h]$, and with $\beta[h]$ the prefix computed by ERV/fly$[h]$. We first establish the correctness of the modified algorithm, and then relate the size of the computed prefix to the informedness of the heuristic.

Theorem 4. *Let h be a safely pruning heuristic function on configurations. Then,* ERV/fly$[h]$ *computes a finite prefix of the unfolding that is complete with respect to the goal marking, and this prefix is contained in that computed by* ERV$[h]$*. Furthermore, if h is admissible, then* ERV/fly$[h]$ *finds an optimal configuration if t_R is reachable. Both claims also hold for* ERV/fly$[\prec, h]$ *where $\prec$ is any semi-adequate order that refines $\prec_h$.*

Proof. ERV/fly$[h]$ is exactly ERV$[h]$ plus two conditions for early termination. As long as neither of these is invoked, ERV/fly$[h]$ behaves *exactly* like ERV$[h]$. If the positive condition (an instance of t_R is dequeued, line 9 in Algorithm 1) is met, t_R is clearly reachable and the the prefix computed by ERV/fly[h] contains a witnessing event. If the negative condition (the h-value of the next event in the queue is ∞, line 6 in Algorithm 1) is met, then the h-value of every event in the queue must be ∞. Since h is safely pruning, this implies none can be extended to a configuration including an instance of t_R. Thus, ERV$[h]$ will not find an instance of t_R either (even though it continues dequeueing these events, inserting them into the prefix and generating successor events until the queue is exhausted). Since ERV$[h]$ is complete, t_R must be unreachable.

As in ERV$[h]$, both claims hold also for any refinement of $\prec_h$. □

If the heuristic h does not assign infinite cost to any configuration, the negative condition can never come into effect and ERV/fly$[h]$ is simply a directed version of McMillan's on-the-fly algorithm. In particular, this holds for $h \equiv 0$.

The next result is that when heuristics are monotonic, improving the informedness of the heuristic can only lead to improved performance, in the sense of a smaller prefix being constructed. In particular, this implies that for any monotonic heuristic h, *the prefix $\beta[h]$ is never larger than that computed by* ERV/fly$[h \equiv 0]$*, regardless of whether the goal transition t_R is reachable or not.* This is not particularly surprising: it is well known in state space search, that – all else being equal – directing the search with a monotonic heuristic cannot result in a larger part of the state space being explored compared to blind search.

In order to compare the sizes of the prefixes computed with two different heuristics, we need to be sure that both algorithms *break ties when selecting events from the queue in a consistent manner*. For a formal definition, consider two instances of ERV/fly: ERV/fly$[h_1]$ and ERV/fly$[h_2]$. We say that a pair of events (e, e') is an inconsistent pair for both algorithms if and only if

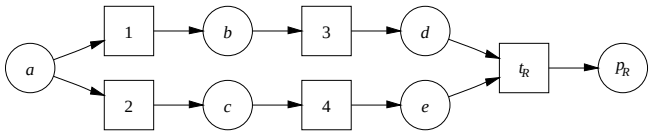

Fig. 2. Example net with an unreachable goal transition (t_R).

1. $[e] \not\prec_{h_i} [e']$ and $[e'] \not\prec_{h_i} [e]$ for $i \in \{1, 2\}$,
2. there was a time t_1 in which e and e' were in the queue of ERV/fly$[h_1]$, and e was dequeued before e', and
3. there was a time t_2, not necessarily equal to t_1, in which e and e' were in the queue of ERV/fly$[h_2]$, and e' was dequeued before e.

We say that ERV/fly$[h_1]$ and ERV/fly$[h_2]$ break ties in a consistent manner if and only if there are no inconsistent pairs between them.

Theorem 5. *If h_1 and h_2 are two monotonic heuristics such that $h_1 \leq h_2$, and* ERV/fly$[h_1]$ *and* ERV/fly$[h_2]$ *break ties in a consistent manner, then every event in $\beta[h_2]$ is also in $\beta[h_1]$.*

Since the all-zero heuristic is monotonic, it follows that the number of events in the prefix computed by ERV/fly$[h]$, for any other monotonic heuristic h, is never greater than the number of such events in the prefix computed by ERV/fly$[h \equiv 0]$, i.e. McMillan's algorithm (although this can, in the worst case, be exponential in the number of reachable states). As noted earlier, for completeness with respect to the goal marking, it is not necessary to insert cut-off events into the prefix (since the marking represented by the local configuration of such an event is already represented by another event in the prefix).

Although the same cannot, in general, be guaranteed for inadmissible heuristics, we demonstrate experimentally below that in practice, the prefix they compute is often significantly smaller than that found by blind ERV/fly, even when the target transition is not reachable. The explanation for this is that all the heuristics we use are safely pruning, which enables us to terminate the algorithm earlier (as soon as the h-value of the first event in the queue is ∞) without loss of completeness.

To illustrate, consider the example net in Figure 2. Suppose initially only place a is marked: at this point, a heuristic such as $h^{\max}$ (defined in the next section) estimates that the goal marking $\{p_R\}$ is reachable in 3 steps (the max length of the two paths). However, as soon as either transition 1 or 2 is taken, leading to a configuration in which either place b or c is marked, the $h^{\max}$ estimate becomes ∞, since there is then no way to reach one of the two goal places.

Pruning safety is a weaker property than admissibility, as it pertains only to a subset of configurations (the dead-end configurations from which the goal is unreachable). Most heuristic functions satisfy it; in particular so do all the specific heuristics we consider in this paper. Moreover, the heuristics we consider

all have equal "pruning power", meaning they assign infinite estimated cost to the same set of configurations. There exist other heuristics, for example those based on pattern databases [20, 21], that have much greater pruning power.

Proof of Theorem 5

Recall that f^* denotes the size of an optimal configuration if one exists, and ∞ otherwise.

Lemma 1. *If h is admissible, all events in $\beta[h]$ have f-value $\leq f^*$.*

Proof. If t_R is not reachable, $f^* = \infty$ and the claim holds trivially. Suppose t_R is reachable. Before the first event corresponding to t_R is dequeued, the queue always contains an event e part of the optimal configuration, which, due to admissibility, has $f([e]) \leq f^*$ (see proof of Theorem 1 (ii)). Thus, the f-value of the first event in the queue cannot be greater than f^*. When an instance of t_R is dequeued, ERV/fly$[h]$ stops. □

Lemma 2. *Let h be a monotonic heuristic. (i) If $e < e'$, i.e. e is a causal predecessor of e', then $f([e]) \leq f([e'])$. (ii) Let β' be any prefix of $\beta[h]$ (i.e. β' is the prefix constructed by* ERV/fly*$[h]$ at some point before the algorithm terminates). If e is an event in β', then every event e' such that $h([e']) < \infty$, $[e'] - \{e'\}$ contains no cut-off event in β' with respect to $\prec_h$, and $[e'] \prec_h [e]$, is also in β'.*

Proof. (i) Consider two events, e and e', such that $e < e'$, i.e. e is a causal predecessor of e'. Since $[e']$ is a finite extension of $[e]$, the definition of monotonicity states that $h([e]) \leq |[e']| - |[e]| + h([e'])$, which implies that $|[e]| + h([e]) \leq |[e']| + h([e'])$, i.e. that $f([e]) \leq f([e'])$. Thus, in any causal chain of events $e_1 < \cdots < e_n$, it holds that $f([e_1]) \leq \cdots \leq f([e_n])$.

(ii) Let e and e' be events such that e is in β', $h([e']) < \infty$, $[e'] - \{e'\}$ contains no cut-off event in β' with respect to $\prec_h$, and $[e'] \prec_h [e]$. We show that e' must be dequeued before e. Since $[e']$ can not contain any cut-off event, other than possibly e' itself, it will be added to the prefix when it is dequeued, because, at this point, e' can not be in the set of recognised cut-off event (the set *cut-off* is only updated on line 12 in the algorithm). Since $e \in \beta'$, this implies that $e' \in \beta'$.

Either e' itself or some ancestor e'' of e' is in the queue at all times before e' is dequeued. By (i), $f([e'']) \leq f([e']) < \infty$ for every causal ancestor e'' of e', and since $|[e'']| < |[e']|$ we have $[e''] \prec_h [e']$ and therefore $[e''] \prec_h [e]$ (by transitivity of $\prec_h$). Thus, all ancestors of e' must be dequeued before e and, since their local configurations contain no cut-off events, added to the prefix. Thus, e' must be put into the queue before e is dequeued, and, since $[e'] \prec_h [e]$, it is dequeued before e. □

Lemma 3. *For any heuristic h, the event e is a cut-off event in prefix β with respect to $\prec_h$ if and only if e is a cut-off event in β with respect to $\prec_m$, where $\prec_m$ is McMillan's order, i.e. $[e] \prec_m [e']$ if and only if $|[e]| < |[e']|$.*

Proof. If e is a cut-off event in β with respect to $\prec_h$, then there exists an event e' in β such that $\text{Mark}([e']) = \text{Mark}([e])$ and $[e'] \prec_h [e]$. The former implies that $h([e']) = h([e])$. The latter implies that either $f([e']) < f([e])$, or $f([e']) = f([e])$ and $|[e']| < |[e]|$. Both imply $|[e']| < |[e]|$ and so $[e'] \prec_m [e]$.

If e is a cut-off event in β with respect to $\prec_m$, then there is e' such that $\text{Mark}([e']) = \text{Mark}([e])$ and $|[e']| < |[e]|$. The former implies that $h([e']) = h([e])$. Therefore, $f([e']) < f([e])$ and so $[e'] \prec_h [e]$. □

Lemma 4. *For any monotonic heuristic h, an event $e \in \beta[h]$ is a cut-off with respect to $\prec_h$ in $\beta[h]$ iff e is a cut-off in the prefix β' built by* ERV/fly*[h] up to the point when e was inserted.*

Proof. That e remains a cut-off event in the final prefix $\beta[h]$ if it was in β' is obvious.

If the h-value of the first event on the queue is ∞, ERV/fly terminates, without inserting the event into the prefix. Thus, since $e \in \beta[h]$, $h([e]) < \infty$.

If e is a cut-off event in $\beta[h]$ with respect to $\prec_h$, there exists an event $e' \in \beta[h]$ such that $\text{Mark}([e']) = \text{Mark}([e])$, $[e'] \prec_h [e]$, and $[e']$ contains no cut-off event. The first two properties of e' are by definition of cut-off events. For the last, suppose $[e']$ contains a cut-off event: then there is another event $e'' \in \beta[h]$, with the same marking and such that $[e''] \prec_h [e']$ (and thus by transitivity $[e''] \prec_h [e]$). If $[e'']$ contains a cut-off event, there is again another event, with the same marking and less according to the order: the recursion finishes at some point because the order $\prec_h$ is well-founded and the prefix $\beta[h]$ is finite. Thus, there is such an event whose local configuration does not contain a cut-off event: call it e'. Consider the prefix $\beta' \oplus \{e\}$ (i.e. the prefix immediately after e was inserted): since it contains e, by Lemma 2(*ii*) it also contains e'. □

Since ERV/fly never inserts into the prefix an event e such that $[e]$ contains an event that is a cut-off in the prefix at that point, it follows from Lemma 4 that if h is a monotonic heuristic, the final prefix $\beta[h]$ built by ERV/fly[h] upon termination contains no event that is the successor of a cut-off event.

Proof (of Theorem 5). Let f_1 and f_2 denote f-values with respect to h_1 and h_2 respectively, i.e. $f_1([e]) = |[e]| + h_1([e])$ and $f_2([e]) = |[e]| + h_2([e])$.

We show by induction on $|[e]|$ that every event $e \in \beta[h_2]$ such that $[e] - \{e\}$ contains no cut-off event in $\beta[h_2]$ with respect to $\prec_{h_2}$, *i.e.*, such that e is not a *post*-cut-off event, is also in $\beta[h_1]$. As noted, by Lemma 4, ERV/fly directed with a monotonic heuristic never inserts any post-cut-off event into the prefix. Thus, it follows from the above claim that every event that may actually be in $\beta[h_2]$ is also in $\beta[h_1]$.

For $|[e]| = 0$ the claim holds because there are no such events in $\beta[h_2]$. Assume that it holds for $|[e]| < k$. Let $e \in \beta[h_2]$ with $[e] - \{e\}$ containing no cut-off events and $|[e]| = k$. By inductive hypothesis, all causal ancestors of e are in $\beta[h_1]$.

Ancestors of e are not cut-off events in $\beta[h_2]$ with respect to $\prec_{h_2}$ (if any of them were e would be a post-cut-off event). Assume some ancestor e' of e is a cut-off event in $\beta[h_1]$ with respect to $\prec_{h_1}$. Then, there is $e'' \in \beta[h_1]$ such

that $\text{Mark}([e'']) = \text{Mark}([e'])$, $[e''] \prec_{h_1} [e']$ and $[e'']$ contains no cut-off event in $\beta[h_1]$ with respect to $\prec_{h_1}$ (by the same reasoning as in the proof of Lemma 4). If some event $e''' \in [e'']$ is a cut-off event in $\beta[h_2]$ with respect to $\prec_{h_2}$, then there exists an event e^4 in $\beta[h_2]$, with equal marking, $[e^4] \prec_{h_2} [e''']$, and such that $[e^4]$ contains no cut-off event. But $|[e^4]| < |[e''']| < |[e'']| < |[e']| < k$, so by the inductive hypothesis, e^4 is also in $\beta[h_1]$, and because $[e^4] \prec_{h_2} [e''']$ implies that $[e^4] \prec_{h_1} [e''']$ (by Lemma 3), this means that e''' is a cut-off event in $\beta[h_1]$ with respect to $\prec_{h_1}$. This contradicts the choice of e'' as an event such that $[e'']$ contains no cut-off events in $\beta[h_1]$ with respect to $\prec_{h_1}$. Therefore, because $|[e'']| < |[e']|$, which implies $[e''] \prec_{h_2} [e']$ (by Lemma 3), it follows from Lemma 2 that e'' is in $\beta[h_2]$. This makes e' a cut-off event in $\beta[h_2]$ with respect to $\prec_{h_2}$, contradicting the fact that e was chosen to be a non-post-cut-off event in $\beta[h_2]$. Thus, no ancestor of e is a cut-off event in $\beta[h_1]$ with respect to $\prec_{h_1}$. It remains to show that e must be dequeued by $\mathsf{ERV/fly}[h_1]$ before it terminates: since ancestors of e are not cut-off events in $\beta[h_1]$ with respect to $\prec_{h_1}$, it follows from Lemma 4 that they are not cut-off events in the prefix built by $\mathsf{ERV/fly}[h_1]$ at that point either, and therefore that, when dequeued, e is inserted into $\beta[h_1]$ by $\mathsf{ERV/fly}[h_1]$.

First, assume that t_R is reachable. By Theorem 4, there is an instance e_R^1 of t_R in $\beta[h_1]$ and an instance e_R^2 of t_R in $\beta[h_2]$ with $|[e_R^1]| = |[e_R^2]| = f^*$. By Lemma 1, $f_2([e]) \leq f_2([e_R^2]) = f^*$ and thus, since $h_1([e]) \leq h_2([e])$, $f_1([e]) \leq f^*$. We do an analysis by cases:

- If $f_1([e]) < f^*$, then $[e] \prec_{h_1} [e_R^1]$ and, by Lemma 2, e is in $\beta[h_1]$.
- If $f_1([e]) = f^*$ and $|[e]| < |[e_R^1]|$, then $[e] \prec_{h_1} [e_R^1]$ and e is in $\beta[h_1]$.
- If $f_1([e]) = f^*$, $|[e]| = |[e_R^1]|$ and $e = e_R^1$, then e is in $\beta[h_1]$.
- $f_1([e]) = f^*$, $|[e]| = |[e_R^1]|$ and $e \neq e_R^1$: e was in the queue of $\mathsf{ERV/fly}[h_1]$ when e_R^1 was dequeued because all causal ancestors of e were in $\beta[h_1]$ at that time (because their f-values are all less than or equal to f^* and the size of their local configurations is strictly smaller). Thus, $\mathsf{ERV/fly}[h_1]$ chose to dequeue e_R^1 before e (and terminated). We show that $\mathsf{ERV/fly}[h_2]$ must have chosen to dequeue e before e_R^1 even though e_R^1 was in the queue of $\mathsf{ERV/fly}[h_2]$, and thus the two algorithms do not break ties in a consistent manner, contradicting the assumptions of the theorem. All causal ancestors e' of e_R^1 satisfy $[e'] \prec_{h_2} [e_R^2]$ and therefore, by Lemma 2, are in $\beta[h_2]$. Hence, when e is dequeued by $\mathsf{ERV/fly}[h_2]$, e_R^1 is in the queue. It cannot be in $\beta[h_2]$ since this would imply termination of $\mathsf{ERV/fly}[h_2]$ before adding e. Thus, $\mathsf{ERV/fly}[h_2]$ chose e over e_R^1.

Next, assume that t_R is unreachable. In this case, $\mathsf{ERV/fly}[h_1]$ can terminate only when the queue is empty or the h-value of the first event in the queue is ∞. The former cannot happen before $\mathsf{ERV/fly}[h_1]$ dequeues e, because all ancestors of e are in $\beta[h_1]$ and thus e was inserted into the queue of $\mathsf{ERV/fly}[h_1]$. Since $e \in \beta[h_2]$, $h_2([e]) < \infty$ (recall that $\mathsf{ERV/fly}$ never inserts an event with infinite h-value into the prefix), and therefore $h_1([e]) < \infty$. Thus, the latter also cannot happen before $\mathsf{ERV/fly}[h_1]$ dequeues e, because e was in the queue of $\mathsf{ERV/fly}[h_1]$ and its h-value is less than ∞. □

4 Heuristics

A common approach to constructing heuristic functions, both admissible and inadmissible, is to define a *relaxation* of the search problem, such that the relaxed problem can be solved, or at least approximated, efficiently, and then use the cost of the relaxed solution as an estimate of the cost of the solution to the real problem, i.e. as the heuristic value [22]. The problem of extending a configuration C of the unfolding into one whose marking includes the target place p_R is equivalent to the problem of reaching p_R starting from $\mathrm{Mark}(C)$: this is the problem that we relax to obtain an estimate of the distance to reach p_R from C.

The heuristics we have experimented with are derived from two different relaxations, both developed in the area of AI planning. The first relaxation is to assume that the cost of reaching each place in a set of places is independent of the others. For a transition t to fire, each place in ${}^\bullet t$ must be marked: thus, the estimated distance from a given marking M to a marking where t can fire is $d(M, {}^\bullet t) = \max_{p \in {}^\bullet t} d(M, \{p\})$, where $d(M, \{p\})$ denotes the estimated distance from M to any marking that includes $\{p\}$. For a place p to be marked – if it isn't marked already – at least one transition in ${}^\bullet p$ must fire: thus, $d(M, \{p\}) = 1 + \min_{t \in {}^\bullet p} d(M, {}^\bullet t)$. Combining the two facts we obtain

$$d(M, M') = \begin{cases} 0 & \text{if } M' \subseteq M \\ 1 + \min_{t \in {}^\bullet p} d(M, {}^\bullet t) & \text{if } M' = \{p\} \\ \max_{p \in M'} d(M, \{p\}) & \text{otherwise} \end{cases} \quad (1)$$

for the estimated distance from a marking M to M'. Equation (1) defines only estimated distances to places that are reachable, in the relaxed sense, from M; the distance to any place that is not is taken to be ∞. A solution can be computed in polynomial time, by solving what is essentially a shortest path problem. We obtain a heuristic function, called $h^{\max}$, by $h^{\max}(C) = d(\mathrm{Mark}(C), \{p_R\})$, where ${t_R}^\bullet = \{p_R\}$. This estimate is never greater than the actual distance, so the $h^{\max}$ heuristic is admissible.

In many cases, however, $h^{\max}$ is too weak to effectively guide the unfolding. Admissible heuristics in general tend to be conservative (since they need to ensure that the distance to the goal is not overestimated) and therefore less discriminating between different configurations. Inadmissible heuristics, on the other hand, have a greater freedom in assigning values and are therefore often more informative, in the sense that the relative values of different configurations is a stronger indicator of how "promising" the configurations are. An inadmissible, but often more informative, version of the $h^{\max}$ heuristic, called h^{sum}, can be obtained by substituting $\sum_{p \in M'} d(M, \{p\})$ for the last clause of Equation (1). h^{sum} dominates $h^{\max}$, i.e. for any C, $h^{\mathrm{sum}}(C) \geq h^{\max}(C)$. However, since the above modification of Equation (1) changes only estimated distances to places that are reachable, in the relaxed sense, h^{sum} is still safely pruning, and in fact has the same pruning power as $h^{\max}$.

The second relaxation is known as the *delete relaxation*. In Petri net terms, the simplifying assumption made in this relaxation is that a transition only

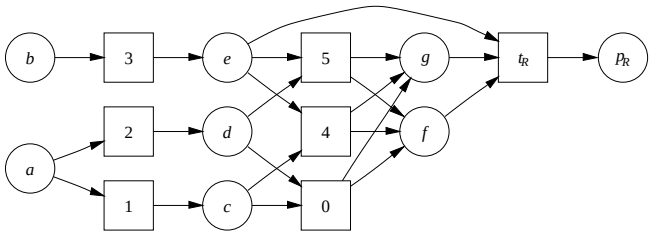

Fig. 3. Relaxed plan graph corresponding to the P/T-net in Figure 1.

requires the presence of a token in each place in its preset, but does not consume those tokens when fired (put another way, all arcs leading into a transition are assumed to be read-arcs). This implies that a place once marked will never be unmarked, and therefore that any reachable marking is reachable by a "short" transition sequence. Every marking that is reachable in the original net is a subset of a marking that is reachable in the relaxed problem. The delete-relaxed problem has the property that a solution – if one exists – can be found in polynomial time. The procedure for doing this constructs a so called "relaxed plan graph", which may be viewed as a kind of unfolding of the relaxed problem. Because of the delete relaxation, the construction of the relaxed plan graph is much simpler than unfolding a Petri net, and the resulting graph is conflict-free[7] and of bounded size (each transition appears at most once in it). Once the graph has been constructed, a solution (configuration leading to p_R) is extracted; in case there are multiple transitions marking a place, one is chosen arbitrarily. The size of the solution to the relaxed problem gives a heuristic function, called h^{FF} (after the planning system FF [9] which was the first to use it). Figure 3 shows the relaxed plan graph corresponding to the P/T-net in Figure 1: solutions include, e.g., the sequences $2, 3, 5, t_R$; $1, 3, 4, t_R$; and $1, 2, 0, 3, t_R$. The FF heuristic satisfies the conditions required to preserve the completeness of the unfolding (in Theorem 1) and it is safely pruning, but, because an arbitrary solution is extracted from the relaxed plan graph, it is not admissible. The heuristic defined by the size of the *minimal* solution to the delete-relaxed problem, known as h^+, is admissible, but solving the relaxed problem optimally is NP-hard [23].

The relaxing assumption of independence of reachability underlying the $h^{\max}$ heuristic is implied by the delete relaxation. This means $h^{\max}$ can also be seen as an (admissible) approximation of h^+, and that $h^{\max}$ is dominated by h^{FF}. However, the independence relaxation can be generalised by considering dependencies between sets of places of limited size (e.g. pairs), which makes it different from the delete relaxation [24].

[7] Technically, delete relaxation can destroy the 1-boundedness of the net. However, the exact number of tokens in a place does not matter, but only whether the place is marked or not, so in the construction of the relaxed plan graph, two transitions marking the same place are not considered a conflict.

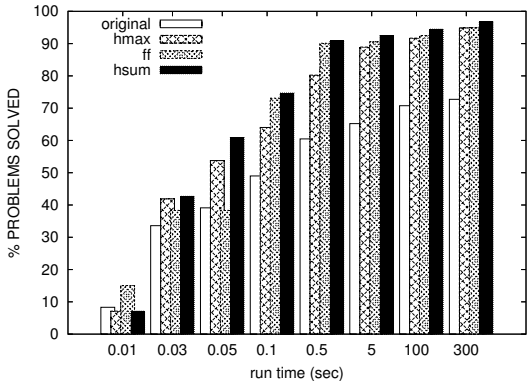

Fig. 4. Results for DARTES Instances

5 Experimental Results

We extended MOLE to use the $\prec_h$ ordering with the $h^{\max}$, h^{sum}, and h^{FF} heuristics. In our experiments below we compare the resulting directed versions of MOLE with the original (breadth-first) version, and demonstrate that the former can solve much larger instances than were previously within the reach of the unfolding technique. We found that the additional tie-breaking comparisons used by MOLE to make the order strict were slowing down all versions (including the original): though they do – sometimes – reduce the size of the prefix, the computational overhead quickly consumes any advantage. (As an example, on the unsolvable random problems considered below, the total reduction in size amounted to less than 1%, while the increase in runtime was around 20%.) We therefore disabled them in all experiments.[8] Experiments were conducted on a Pentium M 1.7GHz with a 2Gb memory limit. The nets used in the experiments can be found at `http://rsise.anu.edu.au/~thiebaux/benchmarks/petri`.

5.1 Petri Net Benchmarks

First, we tested directed MOLE on a set of standard Petri net benchmarks representative of Corbett's examples [25]. However, in all but two of these, the blind version of MOLE is able to decide the reachability of any transition in a matter of seconds. The two problems that presented a challenge are DARTES, which models the communication skeleton of an Ada program, and DME12. [9]

DARTES is the one where heuristic guidance shows the greatest impact. Lengths of the shortest firing sequences required to reach each of the 253 transitions in this problem reach over 90 events, and the breadth-first version could

[8] Thus, our breadth-first MOLE actually implements McMillan's ordering [1].

[9] It has since been pointed out to us that the DME12 problem is not 1-safe, and thus not suitable for either blind or directed MOLE.

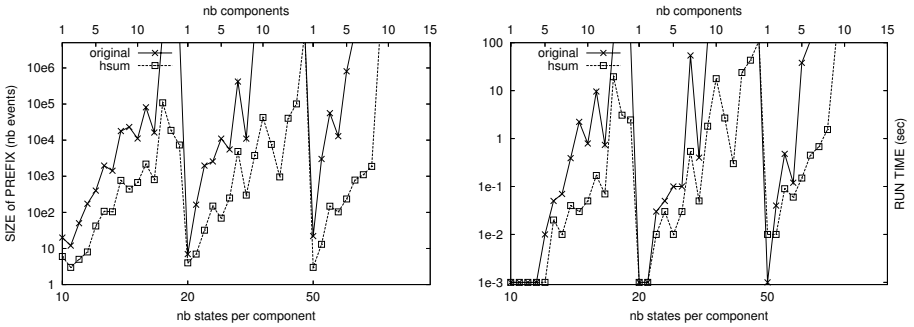

Fig. 5. Results for first set of Random P/T-nets

not solve any instance with a shortest solution length over 60. Overall, the undirected version is able to decide 185 of the 253 instances (73%), whereas the version directed by h^{sum} solves 245 (97%). The instances solved by each directed version is a strict superset of those solved by the original. Unsurprisingly, all the solved problems were positive decisions (the transitions were reachable). Figure 4 presents the percentage of reachability problems decided by each version of MOLE within increasing time limits. The breadth-first version is systematically outperformed by all directed versions.

In the DME12 benchmark, blind MOLE finds solutions for 406 of the 588 transitions, and runs out of memory on the rest. Solution lengths are much shorter in this benchmark: the longest found by the blind version is 29 steps. Thus, it is more difficult to improve over breadth-first search. Nevertheless, MOLE directed with $h^{\max}$ solves an additional 26 problems, one with a solution length of 37. MOLE with the h^{sum} and h^{FF} performs worse on this benchmark.

5.2 Random Problems

To further investigate the scalability of directed unfolding, we implemented our own generator of random Petri nets. Conceptually, the generator creates a set of component automata, and connects them in an acyclic dependency network. The transition graph of each component automaton is a sparse, but strongly connected, random digraph. Synchronisations between pairs of component automata are such that only one (the dependent) automaton changes state, but can only do so when the other component automaton is in a particular state. Synchronisations are chosen randomly, constrained by the acyclic dependency graph. Target states for the various automata are chosen independently at random. The construction ensures that every choice of target states is reachable. We generated random problems featuring $1 \ldots 15$ component automata of 10, 20, and 50 states each. The resulting Petri nets range from 10 places and 30 transitions to 750 places and over 4,000 transitions.

Results are shown in Figure 5. The left-hand graph shows the number of events pulled out of the queue. The right-hand graph shows the run-time. To

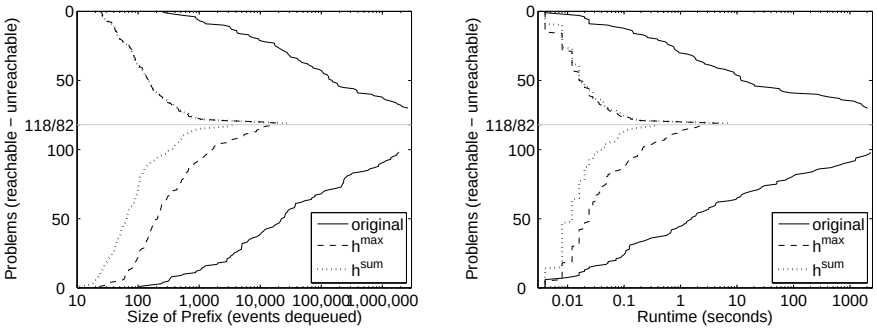

Fig. 6. Results for second set of Random P/T-nets

avoid cluttering the graphs, we show only the performance of the worst and best strategy, namely the original one, and h^{sum}. Evidently, directed unfolding can solve much larger problems than blind unfolding. For the largest instances we considered, the gap reached over 2 orders of magnitude in speed and 3 in size. The original version could merely solve the easier half of the problems, while directed unfolding only failed on 6 of the largest instances (with 50 states per component).

In these problems, optimal firing sequences reach lengths of several hundreds events. On instances which we were able to solve optimally using $h^{\max}$, h^{FF} produced solutions within a couple transitions of the optimal. Over all problems, solutions obtained with h^{sum} were a bit longer than those obtained with h^{FF}.

With only a small modification, viz. changing the transition graph of each component automaton into a (directed) tree-like structure instead of a strongly connected graph, the random generator can also produce problems in which the goal marking has a fair chance of being unreachable. To explore the effect of directing the unfolding in this case, we generated 200 such instances (each with 10 components of 10 states per component), of which 118 turned out to be reachable and 82 unreachable, respectively. Figure 6 shows the results, in the form of distribution curves (prefix size on the left and run-time on the right; note that scales are logarithmic). The lower curve is for solvable problems, while the upper, "inverse" curve, is for problems where the goal marking is not reachable. Thus, the point on the horizontal axis where the two curves meet on the vertical is where, for the hardest instance, the reachability question has been answered.

As expected, h^{sum} solves instances where the goal marking is reachable faster than $h^{\max}$, which is in turn much faster than blind unfolding. However, also in those instances where the goal marking is not reachable, the prefix generated by directed unfolding is significantly smaller than that generated by the original algorithm. In this case, results of using the two heuristics are nearly indistinguishable. This is due to the fact that, as mentioned earlier, their pruning power (ability to detect dead end configurations) is the same.

5.3 Planning Benchmarks

To assess the performance of directed unfolding on a wider range of problems with realistic structure, we also considered some benchmarks from the 4th International Planning Competition. These are described in PDDL (the Planning Domain Definition Language), which we translate into 1-bounded P/T-nets as explained in [4]. Note that runtimes reported below do not include the time for this translation.

The top two rows of Figure 7, show results for 29 instances from the IPC-4 domain AIRPORT (an airport ground-traffic control problem) and 30 instances from the IPC-4 domain PIPESWORLD (a petroleum transportation problem), respectively. The corresponding Petri nets range from 49 places and 18 transitions (AIRPORT instance 1) to 3,418 places and 2,297 transitions (AIRPORT instance 28). The length of optimal solutions, where known, range from 8 to over 160.

Graphs in the first and second columns show cumulative distributions of the number of dequeued events and runtime, respectively, for four different configurations of MOLE: using no heuristic (i.e. $h \equiv 0$), $h^{\max}$, h^{FF} and h^{sum}. Evidently, directed unfolding is much more efficient than blind, in particular when using the inadmissible h^{FF} and h^{sum} heuristics. The original version of MOLE fails to solve 9 instances in the AIRPORT domain, running out of either time (600 seconds) or memory (1Gb), while MOLE with $h^{\max}$ solves all but 4 and MOLE with h^{FF} and h^{sum} all but 2 instances (only one instance remains unsolved by all configurations). In the PIPESWORLD domain, blind MOLE solves only 11 instances, while guided with h^{FF} it solves all but 1.

Graphs in the last column compare the runtimes of the two faster, suboptimal, MOLE configurations with three domain-independent planning systems that are representative of the state-of-the-art. Note that these planning systems implement many sophisticated techniques besides heuristic search guidance. Also, all three are *incomplete*, in the sense that they are not guaranteed to find a solution even when one exists. While the directed unfolder is generally not the fastest, it is not consistently the slowest either. Moreover, with the h^{FF} heuristic, MOLE is very good at finding short solutions in the PIPESWORLD domain: in 14 of the 30 instances it finds solutions that are shorter than the best found by any suboptimal planner that participated in the competition, and only in 1 instance does it find a longer solution. In the AIRPORT domain, all planners find solutions of the same length.

The last row of Figure 7 shows results for an encoding of the OPEN STACKS problem (a production scheduling problem) as a planning problem. A different encoding of this problem (which disabled all concurrency) was used in the 5th planning competition. The corresponding Petri nets all have 65 places and 222 transitions, but differ in their initial markings. Optimal solution lengths vary between 35 and 40 firings. This is an optimisation problem: solving it optimally is NP-complete [26], but only finding any solution is quite trivial. We include this benchmark specifically to illustrate that restricting search to optimal solutions can be very costly. The gap between suboptimal and optimal length unfolding is spectacular: MOLE using the h^{sum} heuristic consistently spends around 0.1

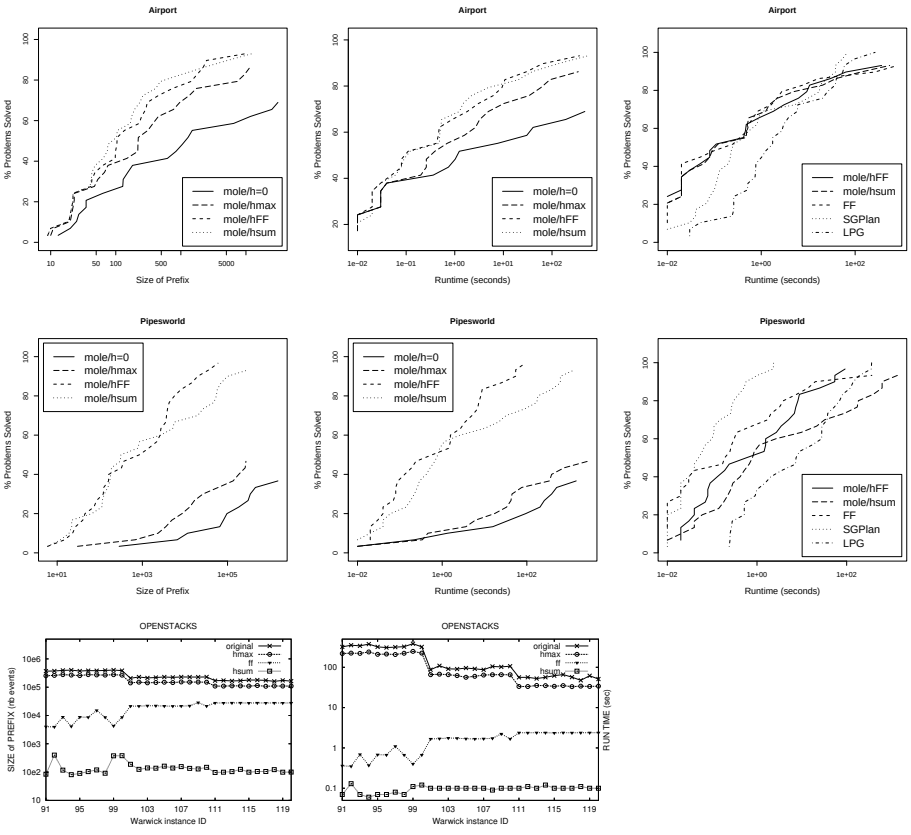

Fig. 7. Results for Planning Benchmarks AIRPORT (top row) and PIPESWORLD (middle row), and the OPENSTACKS problem (bottom row).

seconds solving each problem, while with the admissible $h^{\max}$ heuristic or no heuristic at all it requires over 50 seconds. This shows that directed unfolding, which unlike breadth-first search is not confined to optimal solutions, can exploit the fact that non-optimal OPENSTACKS is an easy problem.

6 Conclusion, Related and Future Work

We have described directed unfolding, which incorporates heuristic search straight into an on-the-fly reachability analysis technique specific to Petri nets. We proved that the ERV unfolding algorithm can benefit from using heuristic search strategies, whilst preserving finiteness and completeness of the generated prefix. Such strategies are effective for on-the-fly reachability analysis, as they significantly reduce the prefix explored to find a desired marking or to prove that none exists. We demonstrated that suitable heuristic functions can be automatically extracted from the original net. Both admissible and non-admissible heuristics can

be used, with the former offering optimality guarantees. Experimental results show that directed unfolding provides a significant performance improvement over the original breadth-first implementation of ERV featured in MOLE.

Edelkamp and Jabbar [27] recently introduced a method for directed model-checking Petri nets. It operates by translating the deadlock detection problem into a metric planning problem, solved using off-the-shelf heuristic search planning methods. These methods, however, do not exploit concurrency in the powerful way that unfolding does. In contrast, our approach combines the best of heuristic search and Petri net reachability analysis. Results on planning benchmarks show that directed unfolding with inadmissible heuristics is competitive (in the sense of not being consistently outperformed) with some of the current state-of-the-art domain-independent planners.

The equivalent of read-arcs is a prominent feature of many planning problems. In our translation to Petri nets, these are represented by the usual "consume-and-produce" loop, which forces sequencing of events that read the same place and thus may reduce the level of concurrency (although this does not happen in the two domains we used in our experiments; they are exceptional in that respect). We believe that a treatment of read-arcs that preserves concurrency, such as the use of place replication [28], is essential to improve the performance of directed unfolding applied to planning in the general case, and addressing this is a high priority item on our future work agenda.

In this paper we have measured the cost of a configuration C by its cardinality, i.e. $g(C) = |C|$. Or similarly, $g(C) = \sum_{e \in C} c(e)$ with $c(e) = 1 \ \forall e \in E$. These results extend to transitions having arbitrary non-negative cost values, i.e. $c : E \rightarrow \mathbb{R}$. Consequently, using any admissible heuristic strategy, we can find the minimum cost firing sequence leading to t_R. As in the cardinality case, the algorithm is still correct using non-admissible heuristics, but does not guarantee optimality. The use of unfolding for solving optimisation problems involving cost, probability and time, is a focus of our current research.

We also plan to use heuristic strategies to guide the unfolding of higher level Petri nets, such as coloured nets [29]. Our motivation, again arising from our work in the area of planning, is that our translation from PDDL to P/T-nets is sometimes the bottleneck of our planning via unfolding approach [4]. Well developed tools such as PUNF[10] could be adapted for experiments in this area.

Acknowledgements It was our colleague Lang White who first recognised the potential of exploring the connections between planning and unfolding-based reachability analysis; we are very thankful and much endebted to him. Many thanks to several of the UFO-07 participants for very insightful discussions. In particular, thanks to Eric Fabre, Victor Khomenko, and Walter Vogler, whose comments helped to significantly improve this paper. The TopNoc reviewers also provided insightful comments, for which we are thankful. Thanks also to Jussi Rintanen and John Slaney for their help with various aspects of this work at some stage or another, and to Stefan Schwoon for his help with MOLE. The authors

[10] `http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/tools.html`

thank NICTA and DSTO for their support via the DPOLP project. NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the ARC.

References

1. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: Computer Aided Verification, 4th International Workshop (CAV'92). Volume 663 of Lecture Notes in Computer Science., Springer (1992) 164–177
2. Esparza, J.: Model checking using net unfoldings. Science of Compututer Programming **23**(2–3) (1994) 151–195
3. Benveniste, A., Fabre, E., Jard, C., Haar, S.: Diagnosis of asynchronous discrete event systems, a net unfolding approach. IEEE Trans. on Automatic Control **48**(5) (2003) 714–727
4. Hickmott, S., Rintanen, J., Thiébaux, S., White, L.: Planning via Petri net unfolding. In: Proc. of 20th Int. Joint Conference on Artificial Intelligence, AAAI Press (2007) 1904–1911
5. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan's unfolding algorithm. Formal Methods in System Design **20**(3) (2002) 285–310
6. Esparza, J., Schröter, C.: Unfolding based algorithms for the reachability problem. Fundamentia Informatica **46** (2001) 1–17
7. Esparza, J., Kanade, P., Schwoon, S.: A negative result on depth first unfolding. Software Tools for Technology Transfer (2007)
8. Bonet, B., Geffner, H.: Planning as heuristic search. Artificial Intelligence **129**(1–2) (2001) 5–33
9. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14** (2001) 253–302
10. McDermott, D.: Using regression-match graphs to control search in planning. Artificial Intelligence **109**(1–2) (1999) 111–159
11. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (1989) 541–580
12. Chatain, T., Khomenko, V.: On the well-foundedness of adequate orders used for construction of complete unfolding prefixes. Information Processing Letters **104** (2007) 129–136
13. Khomenko, V., Koutny, M.: Towards an efficient algorithm for unfolding Petri nets. In: Proc. of 12th Int. Conf. on Concurrency Theory, Springer: LNCS 2154 (2001) 366–380
14. Cheng, A., Esparza, J., Palsberg, J.: Complexity results for 1-safe nets. In: Proc. 13th Conf. on the Foundations of Software Technology and Theoretical Computer Science, Springer: LNCS 761 (1993) 326–337
15. Melzer, S.: Verifikation Verteilter Systeme Mittels Linearer - und Constraint-Programmierung. PhD thesis, Technische Universität München (1998)
16. Heljanko, K.: Using Logic Programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. In: Proc. of 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Springer: LNCS 1579 (1999) 240–254
17. Khomenko, V., Koutny, M.: LP deadlock checking using partial order dependencies. In: Proc. of 11th Int. Conf. on Concurrency Theory, Springer: LNCS 1877 (2000) 410–425

18. Edelkamp, S., Lluch-Lafuente, A., Leue, S.: Directed explicit model checking with HSF-SPIN. In: Proc. of 8th Int. SPIN Workshop. Volume 2057 of Lecture Notes in Computer Science., Springer (2001) 57–79
19. Khomenko, V., Koutny, M., Vogler, W.: Canonical prefixes of Petri net unfoldings. Acta Informatica **40**(2) (2003) 95–118
20. Edelkamp, S.: Planning with pattern databases. In: Proc. 6th European Conf. on Planning, Springer: LNCS (2001) 13–24
21. Haslum, P., Bonet, B., Geffner, H.: New admissible heuristics for domain-independent planning. In: Proc. 20th National Conf. on Artificial Intelligence, AAAI Press / MIT Press (2005) 1163–1168
22. Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley (1984)
23. Bylander, T.: The computational complexity of propositional STRIPS planning. Artificial Intelligence **69**(1–2) (1994) 165–204
24. Haslum, P., Geffner, H.: Admissible heuristic for optimal planning. In: Proc. 6th International Conf. on Artificial Intelligence Planning and Scheduling, Breckenridge, CO, AAAI Press (2000) 140–149
25. Corbett, J.C.: Evaluating deadlock detection methods for concurrent software. IEEE Trans. on Software Engineering **22**(3) (1996)
26. Linhares, A., Yanasse, H.H.: Connection between cutting-pattern sequencing, VLSI design and flexible machines. Computers & Operations Research **29** (2002) 1759–1772
27. Edelkamp, S., Jabbar, S.: Action planning for directed model checking of Petri nets. Electronic Notes Theoretical Computer Science **149**(2) (2006) 3–18
28. Vogler, W., Semenov, A., Yakovlev, A.: Unfolding and finite prefix for nets with read arcs. In: Proc. of 9th Int. Conf. on Concurrency Theory, Springer: LNCS 1466 (1998) 501–516
29. Khomenko, V., Koutny, M.: Branching processes of high-level Petri nets. In: Proc. of 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Springer: LNCS (2003) 458–472