Directed Unfolding of Petri Nets

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

[1]Departamento de Computación, Universidad Simón Bolívar, 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 decisions to 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. 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 reason 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 any particular search strategy. Typically, it has been implemented as a breadth-first search, using the length of paths as the primary means to select the next node to add and to determine cut-off events. The MOLE unfolding tool[1] follows this strategy.

Of the various unfolding-based reachability techniques, experimental results indicate on-the-fly analysis to be most efficient for identifying a single, reachable 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. It has not been obvious what other

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

strategies could be used in the ERV algorithm; recent results have shown depth-first search is incorrect [7]. In this paper, we investigate *directed unfolding*, an approach 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, information about the goal marking can be put to good use 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, heuristics are estimates of the shortest distance from one state to another. Directed unfolding implements a search strategy which explores choices in increasing order of estimated distance to the target marking, as given by the heuristic. 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 a weaker notion of semi-adequate ordering.

Techniques for automatically extracting suitable heuristics from the representation of a transition system and using them to guide search, have 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 actual shortest distances) then directed unfolding finds the *shortest* path to the target marking, just like breadth-first search. Using inadmissible heuristics, completeness and correctness are preserved, and performance is often dramatically improved at the expense of optimality. Alltogether, directed unfolding can solve much larger problems than the original breadth-first ERV algorithm. Moreover, its implementation requires only minor additions to the latter.

The paper is organised as follows. Section 2 provides an overview of place-transition nets, unfolding, 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 we present experimental results covering Petri net benchmarks and Petri net formalisations of automated planning benchmarks. These show that directed unfolding can provide a significant speed up over breadth-first ERV. 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 nodes, but rather captured by explicit variable-event relationships. We consider so-called called *place-transition* (PT) nets, and describe them here only briefly; a detailed expose can be found in [11].

A PT-net consists of a net N and its initial marking M_0. The net is a directed bipartite graph where the nodes are places P and transitions T. 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* M of a PT-net represents the state of the system. It assigns to each place zero or more tokens.

Definition 1. *A PT-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 $M \xrightarrow{t} M'$. We only consider safe (or 1-safe) nets, meaning it is never possible for more than one token to exist in a place. A *firing sequence* $\sigma = t_1, \ldots, t_n$ is a legal sequence of transition firings, i.e. there are markings $M_1, \ldots, M_n$ such that $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} M_n$. This is noted $M_0 \xrightarrow{\sigma} M_n$. A marking M is *reachable* if there is a sequence σ such that $M_0 \xrightarrow{\sigma} M$.

2.2 Unfolding

Unfolding is a method for reachability analysis which exploits and preserves concurrency information in the Petri net. It generates all possible firing sequences of the net, from the initial marking, whilst maintaining a partial order of events based on the causal relation induced by the net.

Unfolding a PT-net produces a pair $\beta = (ON, \varphi)$ where $ON = (B, E, F')$ is an occurrence net, which is a PT-net without cycles, self conflicts nor backward conflicts, and φ is a homomorphism from ON to N that associates the places/transitions of ON with the places/transitions of the PT-net. A node x is in self conflict if there exists two paths to x which start at the same place and immediately diverge. 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 from. Therefore, the process of unfolding involves breaking all reachable places in backward conflict by making independent copies of such places, and thus the ON net may contain multiples copies of the 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.

2.3 Configurations

To understand how an unfolding is built, the most important notions are that of a configuration and local configuration. A *configuration* represents a possible partial run of the net. It is any 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 path from e' to e in ON. Clearly, a configuration C is a fragment of ON such that all events in C can be ordered into a firing sequence with respect to B_0.

A configuration C can be associated with a marking $\mathrm{Mark}(C)$ of the original PT-net by identifying which conditions will contain a token after the events in C are fired from the initial marking; i.e. $\mathrm{Mark}(C) = \varphi((B_0 \cup C^\bullet) \setminus {}^\bullet C)$ 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 PT-net when (only) the events in C occur. The *local configuration* of an event e, denoted $[e]$ is the minimal configuration containing event e. A set of conditions can be simultaneously marked if the union of the local configurations of their presets forms 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 β is infinite and thus cannot be built. However, it is not necessary to build the complete unfolding β, but only a *complete finite prefix* β' of β that contains all the information in β. Formally, a prefix β' of β is *complete* if for every reachable marking M, there exists a configuration $C \in \beta'$ such that $\mathrm{Mark}(C) = M$, and for every transition t enabled by M, there is a configuration $C \cup \{e\}$ such that $e \notin C$ and $\varphi(e) = t$.

The key to obtaining a complete finite prefix is to identify those events at which we can cease unfolding 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]. In the following, $C \oplus \mathcal{E}$ denotes a configuration that extends C with the finite set of events $\mathcal{E}$ disjoint from C.

Definition 2. *A partial order $\prec$ on finite configurations is adequate 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 preserved by finite extensions: if $C_1 \prec C_2$ and $Mark(C_1) = Mark(C_2)$, then for all finite extensions $C_1 \oplus \mathcal{E}_1$ and $C_2 \oplus \mathcal{E}_2$ such that $\mathcal{E}_1$ and $\mathcal{E}_2$ are isomorphicwe have $C_1 \oplus \mathcal{E}_1 \prec C_2 \oplus \mathcal{E}_2$.*

Algorithm 1 The ERV Unfolding Algorithm

Add the conditions in B_0 to the prefix
Initialise the priority queue with the events possible in B_0
Note: the queue is sorted in increasing order wrt $\prec$
while the queue is not empty:
    Remove the first event in the queue (minimal with respect to $\prec$)
    if it is not a cut-off
        Add the event and its postset to the prefix
        Identify the possible next events and insert them in the queue
    endif
endwhile
Add all cut-off events and their postsets to the prefix

Without threat to completeness, we can cease unfolding from an event e, if e takes the net to a marking which can be caused by some other 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]:

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

2.5 ERV Algorithm

Mole is a freeware program which unfolds 1-safe PT-nets. It builds the complete finite prefix following the ERV algorithm depicted in Algorithm 1. Mole uses an adequate order $\prec$ on configurations defined by $C \prec C'$ iff $|C| < |C'|$, further refined into a total order which results in minimal complete prefixes being produced [5]. Note that using this order equates to a breadth-first search strategy.

2.6 On-The-Fly Reachability Analysis

We define the *reachability problem* (also often called coverability problem) for 1-safe PT-nets as follows:

> Reachability: Given a PT-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 [13]. There are various unfolding-based algorithms that decide reachability. Some build the complete finite prefix once and use it to answer multiple reachability questions with e.g. SAT or linear programming [6]. Note that deciding reachability is still NP-complete in the size of the complete finite prefix [6].

In contrast, we are interested in the on-the-fly approach, which experimental results have shown to be most efficient when considering just a single marking [6]. This method was first suggested by McMillan [1]. It involves introducing a new transition t_R to the original net, such that ${}^\bullet t_R = P'$. The net is then unfolded, as described previously, but stops when an event e_R, such that $\varphi(e_R) = t_R$, is retrieved from the queue.[2] At this point we can conclude the set of places P' is reachable. If no such event is identified, the complete finite prefix will be generated, indicating that P' is not reachable.

3 Directing the Unfolding

3.1 Intuitive Idea

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 at a *principled method* of using this information for building the finite prefix in order to turn unfolding into an informed algorithm oriented at solving the reachability task. The resulting approach is called "directed unfolding" as opposed to the standard "blind unfolding"[3].

The basic idea is that for solving 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 events "closer" to t_R as their systematic exploration results in a more efficient search strategy. This approach is only possible if the finite prefix being built is guaranteed to be complete. Otherwise, the algorithm might erroneously conclude that t_R is not reachable. Not every strategy of pulling events out of the queue is sound. Even a simple depth-first unfolding strategy was recently shown to be incorrect in general [7].

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 Definition 2 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.

3.2 Principles

As is standard in state-based search, our orderings are based upon the values of a function f that maps configurations into non-negative numbers. Such function f defines the ordering $\prec_f$ as

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

[2] 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. To guarantee optimality however, even with breadth-first unfolding, it is imperative to wait until the event is pulled out of the queue.

[3] The term "directed" has been used elsewhere to emphasize the informed nature of other model-checking algorithms [14].

Furthermore, the function f is composed of two parts: $f(C) = g(C)+h(C)$ where $g(C) = |C|$ is the number of events in C, and $h(C)$ is any non-negative function such that $h(C) = 0$ if $t_R \in C$ and such that for all pairs of configurations C_1 and C_2 $\text{Mark}(C_1) = \text{Mark}(C_2) \Rightarrow h(C_1) = h(C_2)$. Notice that taking $h \equiv 0$ makes $\prec_f$ into the adequate ordering used by McMillan [1], and that furthermore, by breaking ties appropriately, $\prec_f$ becomes the strict adequate ordering defined in [5] and used in the MOLE implementation of the ERV algorithm.

Following the terminology in heuristic search, the g component is referred to as the cost associated to configuration C while h is referred to as the "heuristic", estimated cost, or distance to reach transition t_R from $\text{Mark}(C)$.

Let us define $h^*(C) = |C'| - |C|$ where $C' \supseteq C$ is the configuration of minimum cardinality that contains t_R if one exists, and ∞ otherwise. 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 $t_R \in C^*$ and C^* is of minimum cardinality among such configurations. We then have,

Theorem 1 (Main). *Let $f(C) = g(C)+h(C)$ and consider the ordering $\prec_f$ as defined above. Then, (i) the ERV algorithm equipped with $\prec_f$ solves* REACHABILITY*, and (ii) it finds an optimal configuration if one exists and h is admissible.*

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 the theorem is that the original ERV algorithm, which equates to taking $h \equiv 0$, finds shortest firing sequences. In the next two sections, we will give examples of heuristic functions, 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.

3.3 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 of the net and consider an infinite sequence of events $e_1 < e_2 < e_3 < \cdots$ 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 $e_1 < e_2 < e_3 < \cdots$ of events there are $i < j$ such that $[e_i] \prec [e_j]$. We thus define

Definition 4. *A partial order $\prec$ on finite configurations is semi-adequate if*

(a) *$\prec$ is well founded, i.e. it has no infinite descending chains,*
(b) *in every infinite chain $C_1 \subset C_2 \subset C_3 \subset \cdots$, there are $i < j$ such that $C_i \prec C_j$, and*

(c) $\prec$ *is preserved by finite extensions: if* $C_1 \prec C_2$ *and* $Mark(C_1) = Mark(C_2)$*, then for all finite extensions* $C_1 \oplus \mathcal{E}_1$ *and* $C_2 \oplus \mathcal{E}_2$ *such that* $\mathcal{E}_1$ *and* $\mathcal{E}_2$ *are isomorphic, we have* $C_1 \oplus \mathcal{E}_1 \prec C_2 \oplus \mathcal{E}_2$*.*

Theorem 2 (Finiteness and Completeness). *If* $\prec$ *is semi-adequate, the prefix produced by the ERV algorithm (Algorithm 1) is finite and complete.*

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 finitness of the prefix computed by ERV for adequate orderings. That proof has three items: (1) shows that each chain of events in the prefix is finite, (2) shows that for each event in the prefix, its pre- and postset are finite, and (3) shows that there are only finitely many reachable events at each depth of the prefix. The proofs of items (2) and (3) do not rely on condition (b) and can be reused verbatim. The proof of (1) can be replaced by a proof by contradiction as follows. Suppose that an infinite chain $e_1 < e_2 < e_3 < \cdots$ of events exists in the prefix. Each event e_i defines a configuration $[e_i]$ with marking $\text{Mark}([e_i])$, and since the number of markings is finite, there is at least one marking that appears infinitely often in the chain. Let $e'_1 < e'_2 < e'_3 < \cdots$ be an infinite subchain such that $\text{Mark}([e_1]) = \text{Mark}([e_j])$ for all $j > 0$. By condition (b) of semi-adequate orderings, there are $i < j$ such that $[e_i] \prec [e_j]$ that together with $\text{Mark}([e_i]) = \text{Mark}([e_j])$ implies that e_j is 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 $\prec_f$ is semi-adequate follows directly from the observation that $g(C) = |C|$ is a strictly monotone function with respect to set inclusion, i.e. $C \subset C' \Rightarrow g(C) < g(C')$. Additionally, the property that configurations with identical markings have identical h-values is required for condition (c).

Theorem 3 (Semi-Adequacy of $\prec_f$). *For any* g *and* h *satisfying the assumptions,* $\prec_f$ *is a semi-adequate order.*

Proof. That $\prec_f$ is transitive follows directly from its definition. For well-foundedness, let $C_1 \succ_f C_2 \succ_f \cdots$ be an infinite descending chain of *finite* configurations with markings $M_1, M_2, \ldots$ respectively. Clearly, not all C_i can be such that $f(C_i) = \infty$ since this would imply $\infty > |C_1| > |C_2| > \cdots \geq 0$ which is impossible. Let $C'_1 \succ_f C'_2 \succ_f \cdots$ be the subchain where $f(C'_i) < \infty$ for all $i > 0$, and $M'_1, M'_2, \ldots$ be the corresponding markings. Since the number of makings is finite, we can extract a further subsubchain $C''_1 \succ_f C''_2 \succ_f \cdots$ such that $\text{Mark}(C''_1) = \text{Mark}(C''_j)$ for all $j > 0$, and thus $h(C''_1) = h(C''_j)$ for all $j > 0$. Therefore, $g(C''_1) > g(C''_2) > \cdots \geq 0$ which is impossible since $g(C) = |C|$ and all C''_i's are finite.

For condition (b), let $C_1 \subset C_2 \subset \cdots$ be an inifinite chain of finite configurations with markings $M_1, M_2, \ldots$ respectively. As before, since the number of marking is finite, there is a subchain $C'_1 \subset C'_2 \subset \cdots$ whose markings are all equal, and thus $h(C'_1) = h(C'_j)$ for all $j > 0$. On the other hand, since all configurations are finite and g is strictly monotone, $0 \leq g(C_1') < g(C_2') < \infty$. If $h(C_1') = \infty$, then $C_1' \prec_f C_2'$, otherwise $f(C_1') = g(C_1') + h(C_1') < g(C_2') + h(C_2') = f(C_2') < \infty$ and $C_1' \prec_f C_2'$.

Finally, if $C_1 \prec_f C_2$ have same markings and the extensions $\mathcal{E}_1$ and $\mathcal{E}_2$ are isomorphic, the extensions $C_1' = C_1 \oplus \mathcal{E}_1$ and $C_2' = C_2 \oplus \mathcal{E}_2$ also have the same markings, and is straghtforward to show that $C_1' \prec_f C_2'$. □

We are now in a position to prove the main theorem. The fact that ERV equipped with $\prec_f$ solves Reachability follows directly from Theorems 2 and 3. It remains to show that if h is admissible and t_R is reachable, then ERV finds an *optimal* configuration.

Proof (of Theorem 1 (ii)). For a proof by contradiction, assume that the configuration $[e_R]$ for the first event e_R found by ERV is not optimal. Observe that the queue always contains an event e such that $[e]$ is a prefix of an optimal configuration C^* (by induction since it holds at the beginning and remains so after each iteration of ERV). Let e be such an event in the queue when ERV pulls e_R out of the queue. We have that

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

since $C^* \supseteq [e]$ is the smallest configuration containing t_R. On the other hand, $[e_R]$ being non-optimal, $f([e_R]) = |[e_R]| > |C^*|$. Thus, $f([e_R]) > f([e])$ and so e_R could not have beed pulled out of the queue. □

3.4 Size of the Finite Prefix

Up to now, we have been mainly concerned with the case when t_R is reachable. Next, we discuss some results related to the size of the prefix generated in case it is not. For this, we need an additional property of the heuristic function h: we say that h is *safely pruning* iff $h(C) = \infty$ implies that there is no configuration $C' \supseteq C$ with $t_R \in C'$, i.e. $h^*(C) = \infty$. This is like admissibility, but applied only to dead end states. If h has this property, then the unfolding can be stopped as soon as the f-value of the best event retrieved from the queue is ∞, since this implies that t_R is unreachable. Note, however, that the prefix generated at this point is not necessarily complete: it may lack some markings that are reachable but irrelevant for the purpose of reaching t_R.

For a heuristic function h, let ERV(h) be the ERV algorithm directed with h, and let $\beta(h)$ be the prefix built by ERV(h) at termination.

Theorem 4. *Let $f^* = |C^*|$ be the cost of an optimal configuration C^*, and ∞ if no solution exists: (i) if h is admissible, then all events $e \in \beta(h)$ have f-value $\leq f^*$. (ii) If $h_1 \leq h_2$ are two admissible heuristics, then all events $e \in \beta(h_2)$ with $g([e]) + h_2([e]) < f^*$ are also in $\beta(h_1)$. If, in addition, h_1 and h_2 are both monotone*[4] *and ties are broken systematically in the same manner, then $\beta(h_2) \subseteq \beta(h_1)$.*

[4] h is *monotone* iff $h(C) \leq |\mathcal{E}| + h(C \oplus \mathcal{E})|$ for all C and all extensions $C \oplus \mathcal{E}$ of C. Monotonicity implies admissibility, but the converse is not true.

We omit the proof for lack of space.

Corollary 1. *If h is admissible and t_R is not reachable, the prefix $\beta(h)$ is always no greater than the one generated by the ERV algorithm with a breadth-first strategy.*

Proof. Follows from the theorem and the fact that ERV is equivalent to directed unfolding with $h \equiv 0$. $\square$

Pruning safety is a weaker property than (general) admissibility, as it pertains only to a subset of configurations. Most heuristic functions safisfy it; in particular so do all the heuristics we consider in this paper. The above result shows that for unsolvable problems, the prefix generated using an admissible heuristic are guaranteed to be no larger than that generated by the standard ERV algorithm. Below, we show experimentally that it can in fact be significantly smaller, even when the heuristic is not (generally) admissible. What matters is the *pruning power* of the heuristic, i.e., its ability to assign an infinite cost estimate to configurations from which the goal marking is unreachable. The heuristics we consider are all equivalent in this respect, but other admissible heuristics, such as e.g. pattern database (PDB) heuristics, have much greater pruning power.

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 search problem can be solved, or at least approximated, efficiently, and 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 [15]. The problem of extending a configuration C of the unfolding into one that includes the target transition t_R is equivalent to the problem of reaching t_R starting from $\mathrm{Mark}(C)$: this is the problem that we relax to obtain an estimate of the distance to reach t_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 consider each place in the preset of a transition independently 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'. The solution to equation (1) can be computed in polynomial time using dynamic programming. We obtain a

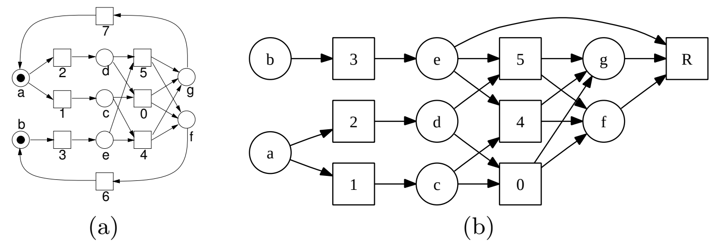

Fig. 1. (a) Example PT-net with marking and (b) corresponding relaxed plan graph.

heuristic function, called $h^{\max}$, by $h^{\max}(C) = d(\mathrm{Mark}(C), {}^{\bullet}t_R)$. This estimate is never greater than the actual distance, so the $h^{\max}$ heuristic is admissible.

In many cases, however, it 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 states. 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 states is a stronger indicator of how "promising" the states 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).

The second relaxation is known as the *delete relaxation*. In Petri net terms, the simplifying assumption made in this relaxation is a transition only 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 is essentially a complete prefix of the unfolding of the relaxed problem. Because of the delete relaxation, the construction of the relaxed plan graph is much simpler than the unfolding of a Petri net, and the resulting graph is conflict-free[5] and of bounded size (each transition appears at most once in it). Once the graph has been constructed, a solution (configuration leading to t_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 1 shows an example of a marked net and the corresponding relaxed plan graph: a minimal solution is the sequence $3, 5, R$; other solutions include, e.g., $1, 3, 4, R$ and $1, 2, 0, 3, R$. The FF heuristic satisfies the conditions required

[5] Technically, delete relaxation can destroy the 1-safeness 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.

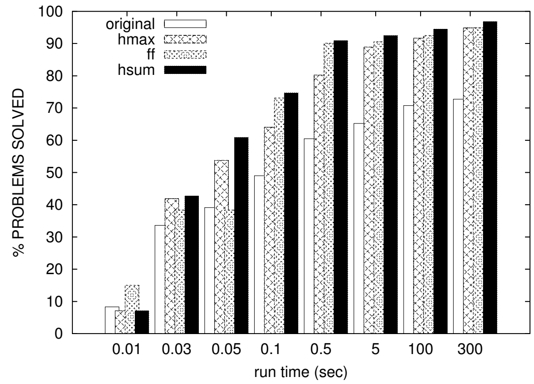

Fig. 2. Results for DARTES Instances

to preserve the completeness of the unfolding (in Theorem 1), 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 [16].

5 Experimental Results

We extended MOLE to use the $\prec_f$ 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[6]. 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

¿From the developers of MOLE we obtained a set of standard Petri net benchmarks representative of Corbett's examples [17]. Only one of them, DARTES, which models the communication skeleton of an Ada program, turned out to be a challenge for MOLE; for other benchmarks in the set, it generates even the complete finite prefix in a matter of seconds.

[6] This means the original version of MOLE in our experiments implements McMillan's ordering [1].

Figure 2 compares the performance of the original version of Mole to the versions directed by each of the heuristics. For each of the 253 Dartes transitions, we recorded the time taken by each version to decide this transition's reachability. The graph shows the percentage of problems solved within increasing time limits, ranging from 0.01 to 300 sec. The original breadth-first version of Mole is systematically outperformed by all of the directed versions. Overall, the original version is able to decide 185 of the 253 problem instances (73%), whereas the version directed by h^{sum} solves 245 of them (97%). The instances solved by each of the directed versions is a strict superset of those solved by the original. Unsurprisingly, all the solved problems were positive decisions (the transitions were reachable). Lengths of shortest solutions to DARTES instances reach up to over 90, and breadth-first could not solve any of the instances that had a shortest solution of length above 60.

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 $1 \dots 15$ 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 4000 transitions.

Results are shown in the top row of Figure 3. The left-hand graph shows the number of events pulled out of the queue. The right-hand graph shows the run-time. To 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 on the unfolding in this case, we generated 200 such instances (each

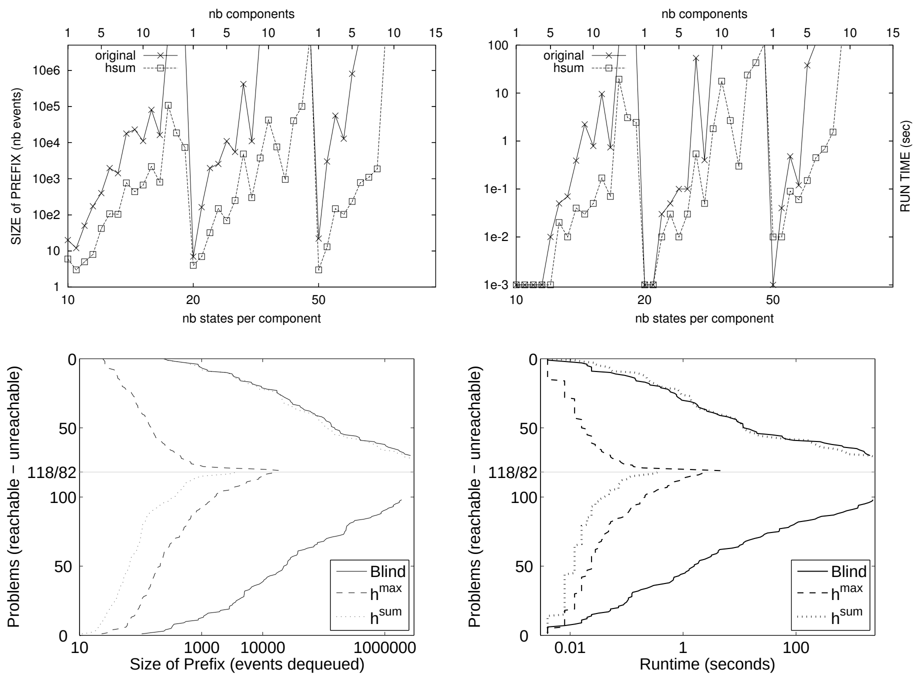

Fig. 3. Results for Random PT-nets

with 10 components of 10 states per component), of which 118 turned out to be reachable and 82 unreachable, respectively. The bottom row of Figure 3 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 problem instance, the reachability question has been answered.

As expected, $h^{\rm sum}$ solves instances where the goal marking is reachable faster than $h^{\rm max}$, 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 benchmarks from the two last editions of the International Planning Competition (IPC-4 and IPC-5). These benchmarks are described in PDDL (the Planning Domain Definition Language), which we translate into 1-safe PT-nets as explained in [4].

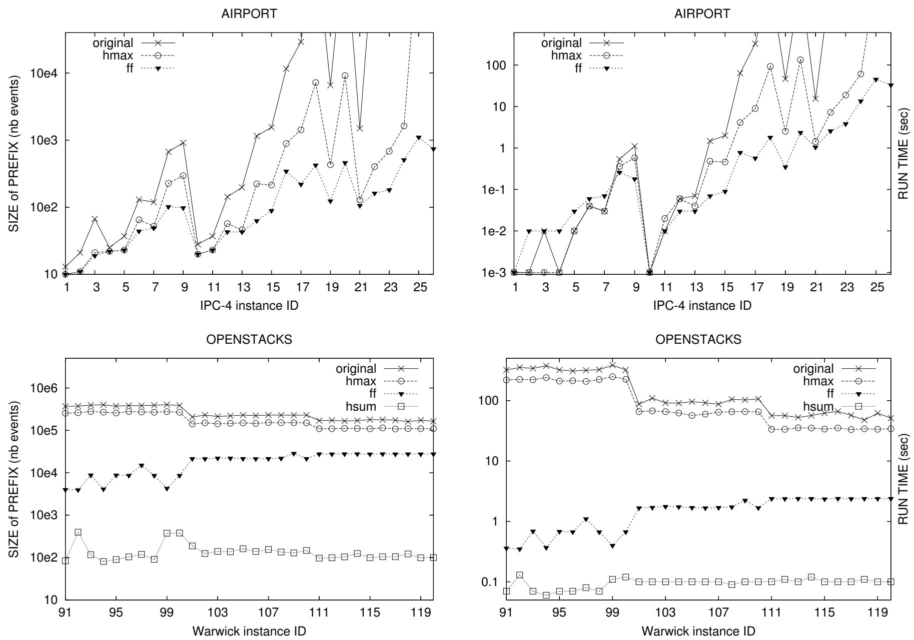

Fig. 4. Results for Planning Problems Airport (top) and Openstacks (bottom)

In the top of Figure 4, we present results for the first 26 IPC-4 instances of Airport, a ground air-traffic control problem. Both the optimal and non-optimal Airport planning problem are known to be PSPACE-complete [18]. The corresponding Petri nets range from 78 places and 18 transitions (instance 1) to 4611 places and 1711 transitions (instance 26). Optimal solution lengths range from 8 to over 200. As before, the left-hand graph shows the number of events pulled out of the queue, and the right-hand graph shows the run-time. To avoid cluttering the graphs, we do not show the performance of h^{sum}. Its curves are comprised between those for h^{FF} and h^{max}. For small instances, the relatively small gain (1 order of magnitude fewer nodes) in unfolding size does not compensate for the overhead incurred in computing the heuristic function. However, for larger instances, directed unfolding reduces both size and run time by over 2 orders of magnitude. The original version of Mole is unable to solve 6 of the instances within a 600 second time limit. These instances describe ground traffic control problems over the topology of half of Munich airport. h^{max} fails to solve the two larger instances, but h^{FF} solves them easily.

In the bottom of Figure 4 we present results for OpenStacks, a production scheduling problem. Optimal OpenStacks is NP-complete [19], while the problem becomes polynomial if optimality is not required. We consider instances Warwick 91-120 which feature 10 products, 10 orders and an increasing ratio of 3 to 5 of products per order. The IPC-5 "propositional" version of OpenStacks disables concurrency. In contrast, while still retaining the IPC-5 optimality criterion, we use the natural encoding of OpenStacks which allows several products to be produced in parallel. The corresponding Petri nets all have 65 places and

222 transitions, but differ in their initial markings. The optimal solution length varies between 35 and 40 operations. In OpenStacks, the gap between directed and breadth-first unfolding is spectacular. The h^{sum} heuristic consistently spend around 0.1 sec solving the problem, that is over 3 orders of magnitude less than the breadth-first version. h^{FF}'s run time ranges from 0.3 sec (instance 91) to 2.8 sec (instance 120). This shows that directed unfolding, which unlike breadth-first search is not confined to optimal solutions, is able to 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 using heuristic search strategies in the ERV unfolding algorithm is safe, in the sense that finiteness and completeness are preserved, and demonstrated that such strategies are effective for on-the-fly reachability analysis, as they significantly reduce the prefix explored to find a desired marking. We showed that heuristic functions automatically extracted from the problem, developed in the area of planning, can be adapted for use with Petri nets.

Hickmott *et. al* [4] showed that if the heuristic is monotone, then $\prec_f$ becomes adequate. In practice, it is difficult to construct an admissible heuristic that is not also monotone. The h^{max} heuristic, and others used in planning, all have this property. We have shown that virtually any heuristic function induces a semi-adequate ordering, which still preserves finiteness and completeness of the generated prefix. If optimality is not required, it is very advantageous to use inadmissible heuristics, as these are in general much more informative and will therefore speed up search. Experimental results demonstrate that directed unfolding provides a significant performance improvement over the original breadth-first implementation of ERV featured in Mole.

Edelkamp and Jabbar [20] 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. The runtimes we obtain in our experiments with planning benchmarks are often competitive with those of the best performing planners. Importantly, this is achieved with an unfolding algorithm which does not handle read-arcs. The treatment of read arcs is essential to improve the performance of directed unfolding applied to planning, and 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. $c : E \to \mathbb{R}$. 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 [21]. Our motivation, again arising from our work in the area of planning, is that our translation from PDDL to PT-nets is sometimes the bottleneck of our planning via unfolding approach [4]. Well developed tools such as PUNF[7] could be adapted for experiments in this area.

Acknowledgements Thanks to Jussi Rintanen for interesting discussions, Stefan Schwoon for his help with MOLE, and to Lang White for suggesting we explore the connections between planning and unfolding-based reachability analysis. The authors 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: CAV. (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 Transactions on Automatic Control **48**(5) (2003) 714–727
4. Hickmott, S., Rintanen, J., Thiébaux, S., White, L.: Planning via Petri net unfolding. In: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07). (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 (To appear)
8. Bonet, B., Geffner, H.: Planning as heuristic search: New results. In: Proceedings of the Ninth International Conference on Automated Planning and Scheduling (ICAPS/ECP-99). (1999) 360–372
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.V.: 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.: A note on the well-foundedness of adequate orders used for truncating unfoldings. Technical Report 998, Newcastle University, School of Computing Science (2007)
13. Cheng, A., Esparza, J., Palsberg, J.: Complexity results for 1-safe nets. In: Proceedings of the Thirteenth Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS-93). (1993) 326–337 LNCS 761.

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

14. Edelkamp, S., Lluch-Lafuente, A., Leue, S.: Directed explicit model checking with hsf-spin. In: Proceedings of the Eighth International SPIN Workshop. (2001) 57–79
15. Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley (1984)
16. Bylander, T.: The computational complexity of propositional strips planning. Artificial Intelligence **69**(1-2) (1994) 165–204
17. Corbett, J.C.: Evaluating deadlock detection methods for concurrent software. IEEE Transactions on Software Engineering **22**(3) (1996)
18. Helmert, M.: New complexity results for classical planning benchmarks. In: Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006). (2006) 52–61
19. Linhares, A., Yanasse, H.: Connection between cutting-pattern sequencing, VLSI design and flexible machines. Computers & Operations Research **29** (2002) 1759 – 1772
20. Edelkamp, S., Jabbar, S.: Action planning for directed model checking of petri nets. Electronic Notes Theoretical Computer Science **149**(2) (2006) 3–18
21. Khomenko, V., Koutny, M.: Branching processes of high-level petri nets. In: Proceedings of the Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-03). (2003) 458–472