Planning Via Petri Net Unfolding: Generalisation and Improvements

Blai Bonet
Dept. de Computación
Universidad Simón Bolívar
Caracas, Venezuela

Patrik Haslum
National ICT Australia &
Australian National University
Canberra, Australia

Sarah Hickmott
National ICT Australia &
University of Adelaide
Adelaide, Australia

Sylvie Thiébaux
National ICT Australia &
Australian National University
Canberra, Australia

Abstract

Recent research has connected automated planning and techniques for analysing reachability in Petri nets. One outcome is a new forward search approach to partial-order planning, based on translating planning problems into Petri nets and guiding a Petri net reachability technique called unfolding with planning heuristics. Unfolding is an attractive method for planning as it naturally enables the recognition and separate resolution of independent subproblems, and can be exponentially more efficient than state-space search. Unfortunately, existing requirements for the soundness and completeness of unfolding have confined the approach to the generation of optimal plans, given additive action costs and monotonic heuristics. In this paper, we establish a relaxation of these requirements under which we are able to generalise the unfolding technique to optimise non-additive criteria such as makespan, and to exploit inadmissible heuristics. This opens the way to using unfolding in the temporal and suboptimal planning settings. We also describe a number of improvements to the translation which reduce the size of the Petri net produced. Our experiments show that planning via unfolding is competitive across a range of problem settings.

Introduction

The past year has witnessed a number of works that effectively exploit the relationship between planning and Petri net analysis (Edelkamp & Jabbar 2006; Hickmott *et al.* 2007; Bonet *et al.* 2007). From the planning perspective, one outcome is a new heuristic forward search method for partial order planning, implemented in the PUP planner (Hickmott *et al.* 2007). PUP translates planning problems into Petri net reachability problems, which are solved using a technique known as *unfolding* (McMillan 1992; Esparza, Römer, & Vogler 2002). The translation preserves the explicit action-variable dependencies present in planning operators. In addition, the concurrency semantics of Petri nets gives the ability to reason about partially ordered sets of actions without considering their interleavings. This is exactly what unfolding does, and for this reason, it can be exponentially more efficient than state-space search.

In Petri net analysis, where the problem is often to prove the absence of deadlocks, unfolding traditionally amounts to a breadth-first search that explores the entire space of partially ordered event sets. An important contribution of Hickmott *et al.* (2007) was to show that the unfolding can be guided towards a goal, using *monotonic* heuristics. Under this guidance, PUP produces cost-optimal non-linear plans for *additive* action costs, i.e., the sum of the costs of the actions in the non-linear plan is minimal. One of the greatest limits of PUP is that additive costs and monotonicity of the heuristic appear to be required to ensure the soundness and completeness of the approach. This confines the unfolding technique to a restricted form of optimal classical planning.

This paper significantly generalises the scope of applicability of the unfolding technique. Our contributions are as follows. We establish relaxed requirements for the soundness and completeness of unfolding, under which even inadmissible heuristics can be used to guide it. This generalisation not only opens the way to using unfolding in a suboptimal planning context, but is also particularly relevant in the context of Petri net reachability analysis, where it scales up to problems beyond reach of existing unfolding tools. Moreover, we extend the unfolding approach beyond the additive cost framework, enabling it to optimise makespan and handle temporal planning problems. We also offer a more efficient translation of planning problems into Petri nets, addressing some of the avoidable causes for explosion. Finally, we show that these extensions, incorporated into PUP, lead to performances that range from honorable to state of the art, depending on the problem setting considered: (sub-)optimal classical planning, optimal temporal planning, and Petri net reachability analysis.

The paper is organised as follows. We start with a brief introduction to Petri nets and unfolding. We go on with a description of our improved translation of planning problems into Petri nets. We then lay the foundations for guiding unfolding using general heuristics, when additive costs are involved. We follow with optimisation of non-additive cost criteria such as of makespan, and its application to temporal planning. Finally, we present experimental results across a range of problem settings, and conclude with a summary of our contribution, related and future work.

Petri Nets and Unfoldings

This section briefly describes the syntax and semantics of Petri nets, presents the reachability problem for Petri nets, and outlines the technique of unfolding. The next section deals with the translation of a planning problem into a suitable Petri net reachability problem.

Place Transition Nets

A place transition (PT) net is a low level Petri net (see left-hand side of Fig. 1). It 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 system. The dynamic behavior is captured by a flow relation, $F : (P \times T) \cup (T \times P) \to \{0, 1\}$, indicating the presence (1) or absence (0) of arcs. The *marking* M of a PT-net represents the state of the system by assigning to each place zero or more tokens, $M : P \to \mathbb{N}$. In this paper we only consider safe (or 1-safe) nets, meaning it is never possible for more than one token to exist in a place.

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 transition $M \xrightarrow{t} M'$ in the 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. A *firing sequence* $\sigma = t_1, \ldots, t_n$ is a legal sequence of transition firings, i.e. there exist markings $M_1, \ldots, M_n$ such that $M_0 \xrightarrow{t_1} \cdots \xrightarrow{t_n} M_n$; denoted as $M_0 \xrightarrow{\sigma} M_n$. A marking M is *reachable* if there is a sequence σ such that $M_0 \xrightarrow{\sigma} M$.

Unfoldings

Unfolding is a method for reachability analysis which 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 $\beta = \langle ON, \varphi \rangle$ where $ON = \langle B, E, F' \rangle$ is an occurrence net, which is a PT-net without cycles or 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 backward conflict is the case 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 φ. 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 right-hand side of Fig. 1 shows a prefix of the unfolding of the PT-net on the left-hand side. Notice the multiple instances of place g 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 is no firing sequence that enables it.

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 $e' \leq e$, and
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. The *local configuration* of an event e, denoted $[e]$ is the minimal configuration containing event e. For instance, in the finite prefix in Fig. 1, $[\mathsf{e5}] = \{\mathsf{e1}, \mathsf{e3}, \mathsf{e4}, \mathsf{e5}\}$.

A configuration C can be associated with a marking $\text{Mark}(C)$ 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. $\text{Mark}(C) = \varphi((B_0 \cup C^\bullet) \setminus {}^\bullet C)$ where $C^\bullet$ (resp. ${}^\bullet C$) is the union of postsets (resp. presets) of all events in C. The marking of C identifies the resultant marking of the original PT-net when (only) the events in C occur. For instance, in Fig. 1, the marking of $[\mathsf{e5}]$ is $\{\mathsf{g}, \mathsf{b}\}$.

Constructing a Finite Prefix

The unfolding process involves identifying which transitions are enabled by the conditions, currently in the occurrence net, that can be simultaneously marked. The identified transitions are referred to as the possible next events. A new instance of each such event is then added to the occurrence net, as are instances of the places in each of 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 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 $\text{Mark}(C) = M$, and for every transition t enabled by M, there exists 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* which are defined with the help of an *adequate order* on configurations (McMillan 1992; Esparza, Römer, & Vogler 2002). 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 1 (Adequate Order) *A partial order $\prec$ on finite configurations is adequate if*

- *$\prec$ is well founded, i.e. it has no infinite descending chains,*
- *if $C \subset C'$ then $C \prec C'$, and*
- *$\prec$ is preserved by finite extensions, i.e. 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 $C_1 \oplus \mathcal{E}_1 \prec C_2 \oplus \mathcal{E}_2$.*

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 transitions (and thus markings) which proceed from e will also proceed from e' (Esparza, Römer, & Vogler 2002).

The ERV Algorithm

A finite complete prefix can be built using the ERV algorithm depicted in Algorithm 1. The algorithm maintains a

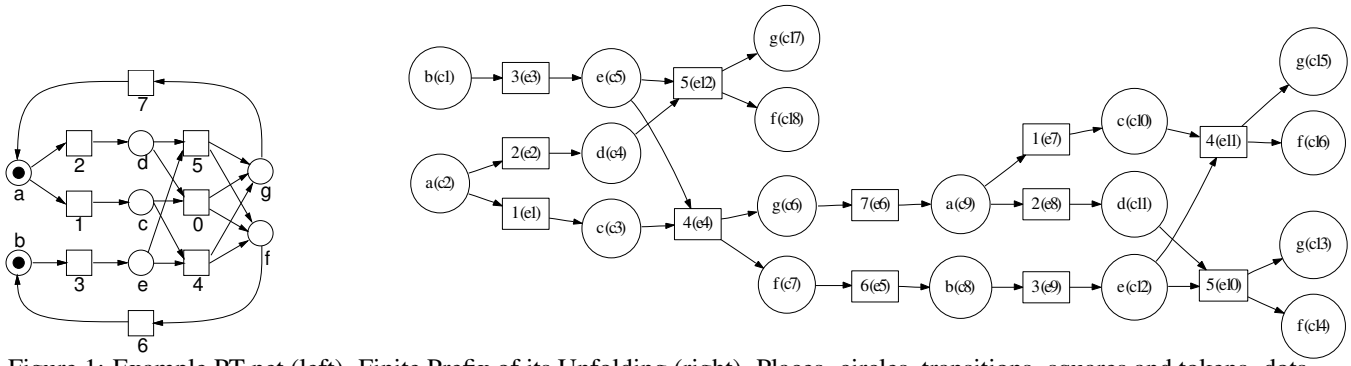

Figure 1: Example PT-net (left). Finite Prefix of its Unfolding (right). Places=circles, transitions=squares and tokens=dots.

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
    if it is not a cut-off
        Add the event and its postset to the prefix
        Insert new possible events into the queue
    endif
endwhile
Add the postsets of all cut-off events to the prefix

priority queue in which the events to be added to the unfolding are ordered in increasing order of $\prec$ (wrt. their local configuration). At each iteration, a minimal event is extracted from the queue and added to the unfolding together with all the conditions in its postset. Additionally, all the events enabled by the new conditions are inserted into the priority queue. The algorithm finishes with a complete unfolding when the queue becomes empty.

The prefix in Fig. 1 is the complete finite prefix that MOLE[1], a publicly available implementation of the ERV algorithm, 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. $\{\mathsf{f}, \mathsf{g}\}$, and each of them is greater than the local configuration of e4 with respect to the adequate partial order implemented by MOLE.

Reachability via Unfoldings

The *reachability problem* for 1-safe PT-nets is the following:

> REACHABILITY: Given a PT-net $\langle P, T, F\rangle$, initial marking M_0, and $P' \subseteq P$; determine whether there is σ such that $M_0 \xrightarrow{\sigma} M$ where $M(p) = 1$ for all $p \in P'$.

This problem is in general PSPACE-complete, yet several algorithms have been designed for it. We are interested in the on-the-fly approach, which experimental results have shown to be most efficient when considering just a single marking (Esparza & Schröter 2001). This method was first suggested by McMillan (1992). 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 generated. At this point we can conclude that the set of places P' is reachable. If no such event is identified, the prefix of the unfolding is generated completely, indicating that P' is not reachable.

Typically, the adequate order used to unfold a PT-net is the partial order induced by cardinality, $C \prec C'$ iff $|C| < |C'|$ (McMillan 1992), or some extensions of it (Esparza, Römer, & Vogler 2002). This amounts to a breadth-first search since the priority queue in such case becomes a FIFO queue.

Planning Problems and Their Translation

A *planning problem* is a tuple $\langle A, I, O, G\rangle$ where A is a set of propositions, $I \subseteq A$ is the initial state, O is a set of grounded operators (actions), and G is a set of goal literals. Operators are characterized by their preconditions $\mathrm{pre}(o)$ and their effects $\mathrm{eff}(o)$, which are both sets of literals.

As usual, a valid sequence of operators for a planning problem is one that maps the initial state to a state satisfying the goal. A non-linear *plan* for the problem is a multiset of operator instances π together with a transitive precedence relation $\leq$ such that any *consistent ordering* of the actions in π with respect to $\leq$ is a valid sequence for the problem.

Overview of the Translation

The reader may now be able to visualise the planning domain in the Petri net world. There is a set of places which represent literals. The presence of a token in a place indicates that the literal is true. The transitions represent the operators. When an operator is executed, the tokens move out and in the relevant places, capturing the change in state.

The automatic translation of a planning problem into a Petri net, as proposed by Hickmott *et al.* (2007), proceeds in two steps, as does our improved translation. The first step transforms all operators into 1-safe operators (defined below) without negative preconditions. The second step translates the modified problem into a Petri net. The 1-safety of the planning operators ensures that the resulting net is also 1-safe, which is essential for the ERV algorithm. Compiling away negative preconditions is necessary because a transition in a Petri net can only be conditioned on the presence of tokens in a place, not their absence. Before translation,

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

the planning problem is subjected to standard preprocessing techniques like reachability and relevance analysis. This reduces the size of the grounded problem, which improves on both the size of the final Petri net and the speed of the translation process. Our improvment targets the first step of the translation; details are presented below.

Ensuring 1-safety

An operator is *1-safe* iff every literal in the operators effects is necessarily false in any reachable state satisfying the operators preconditions. The naive way to ensure that this holds is to include among the preconditions the negation of every literal appearing in the effects. An unsafe operator can thus be converted into collection of safe ones, by creating one "copy" of the operator per combination of values for missing preconditions, and removing, in this new operator, any effect literal contained in the extended precondition (Hickmott *et al.* 2007). However, the number of copies thus created is exponential in the number of missing preconditions, and often it is the case that an operator is safe even though for some of its effects no explicit negation appears in the preconditions. Consider, as an example, an operator for moving a vehicle `?v` between locations `?a` and `?b` in a transportation domain such as Logistics: the effects are `at(?v,?b)` and ¬`at(?v,?a)`, and only the negation of the second, `at(?v,?a)`, is in the operators precondition. But the negation of the first, ¬`at(?v,?b)` is implied by the fact that the two propositions `at(?v,?a)` and `at(?v,?b)` are mutually exclusive, i.e. both cannot hold in any reachable state.

We use standard reachability analysis techniques (computing mutexes and state invariants, as in e.g. (Bonet & Geffner 1999; Helmert 2006)) to try to detect when the negation of a literal in the effects is implied by the literals already present in the preconditions. If this is not the case, we resort to making the negation explicit, creating multiple copies of the operator. The analysis uses polynomial-time approximations of mutual exclusions, so it is still possible that it adds explicit negations that are in fact not needed, but in practice the number of cases where we need to do this are few; many of the standard benchmark domains are 1-safe, or nearly 1-safe, and we are able to detect this.

Eliminating Negative Preconditions

After the problem has been made (or found to already be) 1-safe, negations can be removed following the standard procedure of replacing each negative literal $\neg a$ with a new proposition $\hat{a}$ and modifying all operators so those that make a true make $\hat{a}$ false, and vice versa, thus ensuring that exactly one of a and $\hat{a}$ will hold in every reachable state. (Note that this preserves 1-safety of the problem.) However, even here we can sometimes do better. In many planning problems, there exists already pairs of propositions that are "implicitly" each others negation, and thus we do not need to introduce new propositions for this purpose. A binary mutex is a pair of propositions such that at most one holds in any reachable state: to determine if the two atoms are in fact an implicit negation pair, i.e. if exactly one of them holds in any reachable state, we construct a (hypothetical) state invariant out of the two propositions and run an invariant verification procedure, like that of Helmert (2006).

Conversion into a PT-net

Finally, the 1-safe and negation free planning problem is translated into a Petri net by a straightforward mapping of propositions to places and actions to transitions. Formally, a planning problem $\langle A, I, O, G\rangle$ with safe operators and no negative preconditions is translated into a PT net $\langle P, T, F\rangle$ with initial marking M_0 where

- the places P are the propositions A plus a place R;
- the transitions T are the operators O plus a transition t_R;
- the preset of each transition $o \in O$ is ${}^\bullet o = \text{pre}(o)$, i.e., the set of places in $\text{pre}(o)$;
- the postset of each transition $o \in O$ is $o^\bullet = \text{eff}(o) \cup \{p : p \in \text{pre}(o) \text{ and } \neg p \notin \text{eff}(o)\}$, the places for $\text{eff}(o)$ plus the places for the preconditions that o does not delete;
- the preset of the goal transition is ${}^\bullet t_R = G$, i.e., the set of goal places; it postset is $t_R{}^\bullet = \{R\}$ (goal reached);
- $M_0(p) = 1$ if and only if $p \in I$.

The additional places in the postsets are due to the semantics of Petri nets: unlike a planning operator a transition always consumes its preconditions. Non-deleted preconditions are modeled by having the transition output a token back into the corresponding place. In classical planning, this models the semantics of the planning problem faithfully, in the sense that a plan is valid for the planning problem exactly when the corresponding set of transition firings is a configuration of the Petri net, and their costs are the same. However, in temporal planning the correspondance is not exact, because there are cases in which two operators are allowed to execute concurrently according to the planning semantics (as defined by Smith & Weld 1999), but where the corresponding transitions in the Petri net are not concurrent. This happens only when the operators share a non-deleted precondition. An alternative translation uses a separate place for each "reader" and preserves the planning semantics (Vogler, Semenov, & Yakovlev 1998).

Planning via Unfolding

The planning problem can now be cast as the reachability of t_R in the PT-net generated by translation. This reachability problem is solved using on-the-fly unfolding, as explained earlier. When t_R is reached, its local configuration gives a non-linear plan for the planning problem. The PUP planner is based on this idea and uses the MOLE implementation of the ERV algorithm as the underlying unfolding tool.

However, as pointed out earlier, the standard cardinality based ordering implemented in MOLE leads to a breadth-first search. Hickmott *et al.* (2007) identified that the ordering can be changed in such a way as to implement heuristic search. In particular, provided that the costs of the actions in a plan are additive, any monotonic heuristic yields an adequate order. They also pointed out that non-monotonic, let alone inadmissible heuristics do not fullfill the adequacy requirements, and therefore lead to a potential incompletness of the planner. The next section extends Hickmott's results to arbitrary heuristics, including inadmissible ones.

Directed Unfolding with Additive Costs

As done in heuristic search, we are interested in order relations induced by functions f defined on configurations

which are decomposed in two parts as

$$f(C) \;=\; g(C) + h(C)\,,$$

where $g(C)$ defines the 'cost' of the current configuration C and $h(C)$ 'estimates' the distance from C to transition t_R. To simplify the exposition, we assume that all costs are unit costs; the generalization to arbitrary positive costs is straightforward as just need to replace the expression $|C|$ by $\sum_{e\in C} c(e)$ where $c(e)$ is the cost for event (operator) e. In this case, $g(C) = |C|$ is the number of events in C while $h(C)$ is any non-negative valued function such that $h(C) = 0$ if $t_R \in C$, and $h(C_1) = h(C_2)$ if $\mathrm{Mark}(C_1) = \mathrm{Mark}(C_2)$. Such function f defines an ordering $\prec_f$ given by

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

which is adequate if the heuristic function is monotonic.

In order to define notions of monotonic and admissible heuristics, let us define the optimal heuristic function $h^*(C) = |C'| - |C|$ where $C' \supseteq C$ is the minimum[2] configuration C' that contains t_R if such exists, and ∞ otherwise. Then, h is an *admissible* heuristic if $h(C) \leq h^*(C)$ for all finite configurations C, and h is a *monotonic* heuristic if $h(C) \leq |C'| - |C| + h(C')$ for all finite $C' \supseteq C$. Observe that monotonic heuristics are also admissible. Additionally, let us say that a configuration C^* is *optimal* if $t_R \in C^*$ and C^* is of minimum cardinality among such configurations.

It is not hard to show that monotonic heuristics induce adequate orders (Hickmott *et al.* 2007), but that there is no such guarantee for admissible and non-admissible heuristics. The problem lies in the second condition in Definition 1. However, such condition can be relaxed by requiring that in every sufficiently large chain $C_1 \subset C_2 \subset \cdots \subset C_m$ there are $1 \leq i < j \leq m$ such that $C_i \prec C_j$. The resulting orderings will be referred to as *semi-adequate orderings*. Our first result is that semi-adequate orders can be used to define the cut-off events in a sound manner.

Theorem 2 *Let the order relation $\prec$ be semi-adequate. Then, the ERV algorithm solves* REACHABILITY*; i.e. finds transition t_R if such transition is reachable, or generates a complete finite prefix otherwise.*

Moving from adequate to semi-adequate orders permit to plug admissible and inadmissible heuristics into the search.

Theorem 3 *Let $g(C) = |C|$, h be a non-negative valued function such that $h(C) = 0$ if $t_R \in C$, and $h(C_1) \leq h(C_2)$ if $C_1 \subseteq C_2$ and $\mathrm{Mark}(C_1) = \mathrm{Mark}(C_2)$, and $\prec_f$ defined as above for $f = g + h$. Then, $\prec_f$ is a semi-adequate order and thus ERV solves* REACHABILITY*. Moreover, if h is admissible and t_R is reachable, ERV finds an optimal configuration.*

Directed Unfolding for Makespan

For temporal planning, we set the cost of the actions, and thus the cost of the transitions, to the durations of the operators, i.e. $c(e) = dur(o)$ if event e corresponds to operator o. Since the makespan of a partial plan is the duration of the critical path in the plan, we define the cost of a configuration as the costlier chain in the configuration, i.e. $g(C) = \max_{\sigma\in C} \sum_{e\in\sigma} c(e)$, where the max is over all chains σ in configuration C. Likewise, each place p in the (final) marking of C has a corresponding cost that refer to the time at which p is asserted in the configuration C. In symbols,

$$c(C, p) \;=\; \max_{\sigma_p \in C} \sum_{e\in\sigma_p} c(e)$$

where the max is over all chains σ_p that *support* p. Clearly, $g(C) = \max\{c(C, p) : p \in \mathrm{Mark}(C)\}$. The *optimal makespan heuristic* is then defined as $h^*(C) = \min_{C'} g(C')$ where the min is over all $C' \supseteq C$ that contain t_R.

The heuristic $h(C)$ must estimate the cost of reaching t_R from $\mathrm{Mark}(C)$ while considering the *relative times* at which the places $p \in \mathrm{Mark}(C)$ are asserted. We thus define the relatives times $t(p)$ as $t(p) = c(C, p) - g(C)$. Observe that these quantities are always less than or equal to zero.

We consider a heuristic h^{tmp} similar to the $h^{\max}$ heuristic used in classical planning by means of dynamic programming seeded at the relative times; i.e.

$$h^{\text{tmp}}(C) \;=\; \max\{0, dis(C, {t_R}^\bullet)\}\,,$$

$$dis(C, F) \;=\; \begin{cases} \max_{p\in F} t(p) & F \subseteq M \\ \min_{t\in {}^\bullet p} c(t) + dis(C, {}^\bullet t) & F = \{p\},\ p \notin M \\ \max_{p\in F} dis(C, \{p\}) & \text{otherwise} \end{cases}$$

where $M = \mathrm{Mark}(C)$. It is not hard to show that if $C \subseteq C'$, then $g(C) + h^{\text{tmp}}(C) \leq g(C') + h^{\text{tmp}}(C')$ and therefore, this heuristic is admissible and monotonic.

Unfortunately, in the case of temporal planning, the order $\prec_f$ induced by $f(C) = g(C) + h^{\text{tmp}}(C)$ is not semi-adequate since the third condition in Definition 1 no longer holds. To see this, consider two configurations C and C' with identical markings $\{\mathsf{a}, \mathsf{b}\}$, and makespans $g(C) = 4$ and $g(C') = 5$ given by the assertion times:

$$c(C, \mathsf{a}) = 3\,,\ c(C, \mathsf{b}) = 4\,,\ c(C', \mathsf{a}) = 2\,,\ c(C', \mathsf{b}) = 5\,.$$

Let e be an event with unique preset ${}^\bullet\mathsf{e} = \{\mathsf{a}\}$ and cost 5, and f be the final event t_R with ${}^\bullet\mathsf{f} = \{\mathsf{b}\}$ and cost 1. On one hand we have, $f(C) = g(C) + h^{\text{tmp}}(C) = 4 + 1$ and $f(C') = g(C') + h^{\text{tmp}}(C') = 5 + 1$. On the other hand, for $D = C \cup \{\mathsf{e}\}$ and $D = C' \cup \{\mathsf{e}\}$, $f(D) = 8$ and $f(D') = 7$. Thus, $f(C) < f(C')$ and $f(D) > f(D')$.

As shown, the problem is that subcritical paths in a configuration can become critical once the configuration is extended with an event, and thus the ordering of two configurations can be reversed after the extension. The fix is to require extra conditions when one goes from f to $\prec_f$. Let us define the makespan order $\prec_{ms}$, induced by f, as $C \prec_{ms} C'$ iff

- (i) $f(C) < f(C')$, or $g(C) < g(C')$ if $f(C) = f(C') = \infty$, and
- (ii) if $\mathrm{Mark}(C) = \mathrm{Mark}(C')$, then $c(C, p) \leq c(C', p)$ for all $p \in \mathrm{Mark}(C)$.

In this way, we avoid such inversions and thus the third condition of semi-adequate orderings is restored. Observe that in the case $\mathrm{Mark}(C) = \mathrm{Mark}(C')$, condition (ii) implies condition (i), and so the implementation should check (ii) first. The main result for temporal planning follows.

[2] Cardinality wise.

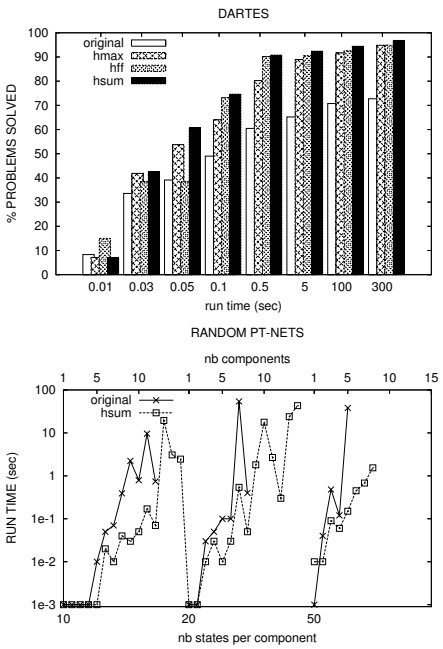

Figure 3: Results for Reachability Analysis on DARTES (top) and on RANDOM PT-nets (bottom)

Theorem 4 *Let h be a non-negative valued function on configurations such that $h(C) = 0$ if $t_R \in C$. Then, the ordering $\prec_{ms}$ induced by $f = g + h$ is semi-adequate and thus ERV solves* REACHABILITY. *Furthermore, if h is admissible, then ERV finds optimal configurations.*

In the next section, we present experimental results of our approach with this ordering on temporal planning problems, with unit and non-unit durations.

Experiments

We extended MOLE, the unfolder underlying PUP, to use the different orderings induced by adaptation of the planning heuristics $h^{\max}$, h^{sum}, and h^{FF} for planning with additive costs, and of the h^{tmp} heuristic for optimising makespan in classical and temporal planning. $h^{\max}$ and h^{sum} are the usual h^1 distance heuristics with $\max$ and sum, and h^{FF} is the FF heuristic. $h^{\max}$ and h^{tmp} are monotonic, the others are inadmissibe. We performed experiments on four types of problems: Petri net reachability problems, suboptimal and optimal classical planning, and optimal temporal planning. Below, we also report results concerning our translation.

Reachability Analysis for Petri Nets

The original version of MOLE is a state of the art tool for Petri net reachability analysis. Here we show that the directed version of MOLE solves problems that were previously beyond reach of its parent.

DARTES is a challenging Petri net benchmark which models the communication skeleton of an Ada program (Corbett 1996). The top graph in Fig. 3 compares the performance of the original version of MOLE to versions directed by the heuristics. We recorded the time taken by each version to decide the reachability of each of the 253 DARTES transitions. The graph shows the percentage of problems solved within increasing computation time limits 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 solves 185 of the 253 problem instances (73%), whereas the version directed by h^{sum} solves 245 of them (97%). The instances solved by each directed version is a strict superset of those solved by the original.

To study the scalability of directed unfolding, we consider random Petri nets obtained by connecting a set of component automata in an acyclic dependency network. The random problems feature $1-15$ 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. The bottom graph in Fig. 3 shows the run times of the original version of MOLE and of the version directed by h^{sum}. Evidently, the latter can solve much larger problems. For the largest instances we considered, the gap reached over 2 orders of magnitude in speed and 3 in the size of the unfolding produced. The original version could merely solve the easier half of the problems.

Domain		av. incr. actions	av. red. size	nb. solved	av. red. time
AIRPORT	old	90%	88%	18	x14
	new	39%		21	
PIPESWORLD	old	900%	89%	10	x20
	new	0%		14	

Table 1: Benefits of the Improved Translation

Translation

Table 1 illustrates the benefits of our improved translation using the IPC4 domains AIRPORT (instances 1-21) and PIPESWORLD (no-tankage, instances 1-30). The table successively reports the average increase in the number of actions required to ensure 1-safety under the old and new translations, the average reduction (new vs old) in size (number of nodes) of the Petri net, the number of problems optimally solved by PUP with the $h^{\max}$ heuristic, and the average reduction (new vs old) in the run-time of PUP.

For PIPESWORLD, the new translation is able to entirely eliminate the need to introduce new operators. In both benchmarks, the reduction in size that follows is important (90%), and the gains in terms of ability to solve problems and run-time (14 and 20 times faster) are significant.

Suboptimal Classical Planning

Turning to planning, we first performed experiments with PUP guided by the inadmissible h^{FF} and h^{sum} heuristics, to evaluate its qualities as a suboptimal planner. Using the h^{sum} heuristic resulted, almost consistently, in worse results, so below we consider only PUP with h^{FF}. For comparison, we ran two other suboptimal planners: LPG-Td (Gerevini, Saetti, & Serina 2006) and SGPlan (Chen, Wah, & Hsu 2006), using the AIRPORT and PIPESWORLD domains. The run-times are presented in the left-hand graphs of Fig 2.

In the AIRPORT domain, PUP and the other planners find plans of the same length, which also equal the best (i.e.

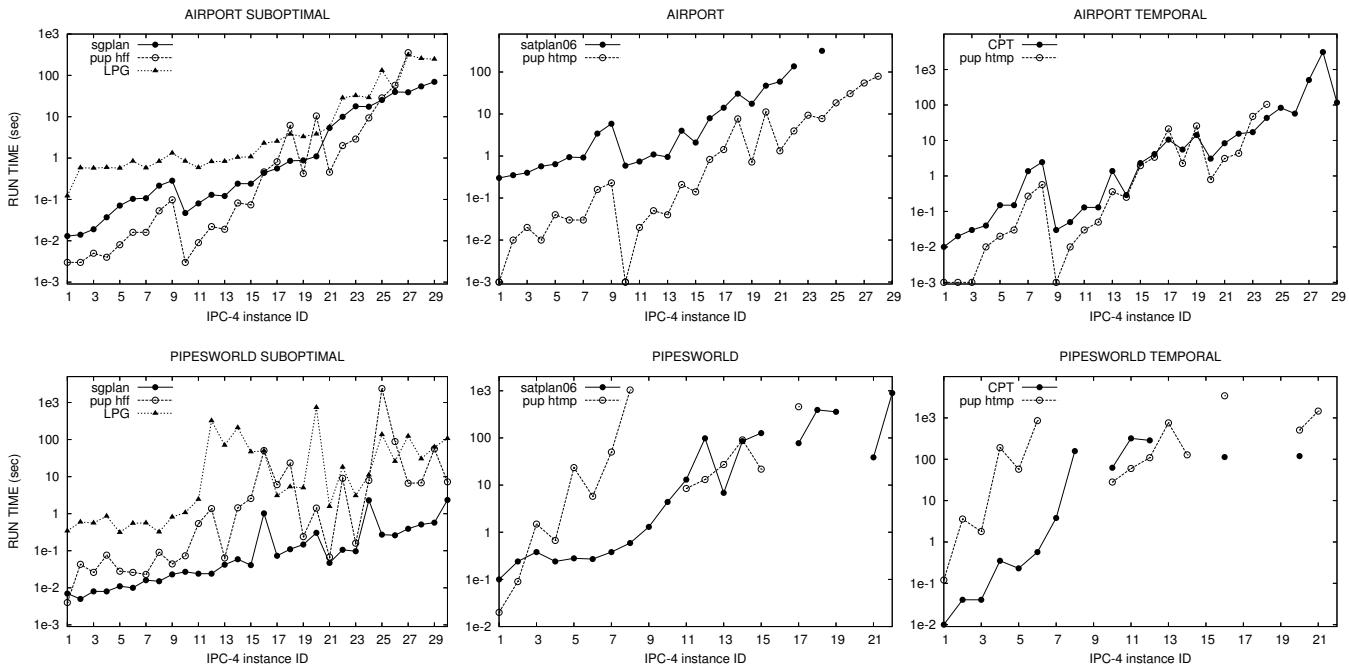

Figure 2: Run Times for Suboptimal Classical Planning (left), Optimal Classical Planning (center), and Optimal Temporal Planning (right). Benchmarks are AIRPORT (top), PIPESWORLD (bottom). Times do not include translation.

shortest) plan length reported by any suboptimal planner in IPC4. We conjecture that these problems have a single, obvious solution that all planners find. For most problems, PUP is faster than LPG-Td and comparable with SGPlan, but on larger problems SGPlan is faster.

Results in PIPESWORLD are more interesting. PUP's runtime is very competitive with LPG-Td and falls short of SGPlan's, but in this domain, PUP finds plans of superior quality, not only to the two planners in the comparison but in many cases shorter than the best result reported by *any* planner in IPC4. See Table 2.

Optimal Classical Planning

Hickmott *et al.* have shown that PUP is competitive with the HSP planner (Haslum 2006) which is the state of the art for optimal classical planning with arbitrary additive costs. Here, we therefore only consider optimal parallel planning, that is, makespan optimisation with unit costs, for which the leading planner is SATPLAN06 (Kautz, Selman, & Hoffmann 2006). The center graphs in Fig. 2 show the run-times of SATPLAN and PUP (run with the h^{tmp} heuristic) on AIRPORT and PIPESWORLD. In both domains, the concurrency in the net generated by our translation is guaranteed to coincide with that allowed in the original planning problem.

PUP clearly outperforms SATPLAN on AIRPORT, solving more problems faster, but is equally clearly outperformed by SATPLAN on PIPESWORLD. These results are not surprising: SATPLAN is well known to excel on problems with short optimal parallel plans, as is the case for PIPESWORLD, while the directed version of PUP can handle longer optimal plans, provided independent subproblems exist.

Optimal Temporal Planning

We compared the temporal version of PUP, using the h^{tmp} heuristic, with the CPT planner (Vidal & Geffner 2006). CPT, like PUP, is optimal w.r.t. makespan, and uses non-directional branching together with constraint propagation techniques for pruning in search. It is likely the best optimal temporal planner, overall, at the moment. We also ran blind PUP and the TP4 planner (Haslum 2006), but since both are completely outperformed by CPT and PUP with h^{tmp} we do not include their results. As shown in the right-hand graphs of Fig 2, the results of CPT are generally better in the AIRPORT domain. In PIPESWORLD, however, results of the two planners are more mixed, with both solving some problems not solved by the other, and both being sometimes faster.

Conclusion, Related, and Future Work

Directed unfolding is a promising approach for planning since it combines the power of heuristic forward search with the ability to reason directly about partially-ordered plans. In this paper, we have presented important extensions and generalisations of this technique. We have shown that it can safely be used in conjunction with inadmissible heuristics for suboptimal planning and Petri net reachability analysis. We have extended it to optimise makespan for temporal planning. Even though PUP does not resort to the sophisticated heuristics and search strategies deployed by state of the art planners, its performance across a range of problem settings is extremely encouraging.

Edelkamp and Jabbar (Edelkamp & Jabbar 2006) recently introduced a method for directed model-checking Petri nets. It operates by translating the deadlock detection problem

id	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
PUP	14	**13**	**18**	20	24	**16**	30	26	**42**	22	30	**26**	**32**	14	**31**	18	**24**	**35**	**39**	**28**	**35**	**33**	**40**
best	**10**	14	20	20	24	20	30	26	70	22	30	28	38	14	35	18	37	42	55	30	47	40	51

Table 2: Plan length reported by PUP with h^{FF} vs best length reported by any suboptimal IPC4 planner (PIPESWORLD)

into a metric planning problem, solved using a heuristic state-space search planner (metric-FF planner (Hoffmann 2003)). Such methods however, do not exploit concurrency in the powerful way that unfolding does. In contrast, our approach combines heuristic search with the best of Petri net reachability analysis.

There is much scope for further developments to the planning via unfolding approach. Our priority for future work is to investigate Petri nets with read-arcs (Vogler, Semenov, & Yakovlev 1998). Such arcs do not consume tokens and are therefore useful to model the accurate semantics of concurrent planning in the presence of shared non-deleted preconditions. This has the potential to greatly improve the performance of PUP, as it will not need to examine the interleaving of events which only conflict on read places. Unfolding with read-arcs is rather complex, and is a hot research topic in Petri net reachability analysis (Baldan *et al.* 2007)

There is also further room for improvement in the translation of planning into Petri nets, which can occasionally be the bottleneck of the approach. We experimented with a translation linear in the number of propositions, but quadratic in the number of actions in the domain as it requires "mutex" places to ensure 1-safety. But we found that in many benchmarks, the number of mutex places greatly dominates the size of the Petri net. This legitimates the translation we and Hickmott *et al.* consider, which is linear when the actions are 1-safe, but is exponential in the number of operators effects in the worst case. However, it would be beneficial to combine the two translations as appropriate.

More ambitious developments concern more compact translations into high-level nets making use of multi-valued variables. The use of heuristic strategies to guide the unfolding of higher level Petri nets, such as coloured nets (Khomenko & Koutny 2003), is also interesting in its own right. Well developed tools such as PUNF[3] could be adapted accordingly for experiments in this area.

Finally, the connections between unfolding and structural and complexity analysis for planning should be studied. This includes e.g. determining whether a precise relationship holds between the size of the unfolding and the width of the causal graph or other structural parameters.

Acknowledgements

Thanks to Stefan Schwoon for his help with MOLE and interesting discussions. Special thanks to Lang White for suggesting we explore the connections between planning and unfolding-based reachability analysis, and discussing optimisation in the unfolding framework. The authors also 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.

[3] http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf

References

Baldan, P.; Corradini, A.; König, B.; and Schwoon, S. 2007. Mcmillan's complete prefix for contextual nets. Submitted for publication.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *Proceedings of the Ninth International Conference on Automated Planning and Scheduling (ICAPS/ECP-99)*, 360–372.

Bonet, B.; Haslum, P.; Hickmott, S.; and Thiébaux, S. 2007. Directed unfolding of petri nets. Submitted to the Workshop on Unfolding and Partial-Order Techniques (UFO-07).

Chen, Y.; Wah, B.; and Hsu, C. 2006. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of AI Research* 26:323–369.

Corbett, J. C. 1996. Evaluating deadlock detection methods for concurrent software. *IEEE Transactions on Software Engineering* 22(3).

Edelkamp, S., and Jabbar, S. 2006. Action planning for directed model checking of petri nets. *Electronic Notes Theoretical Computer Science* 149(2):3–18.

Esparza, J., and Schröter, C. 2001. Unfolding based algorithms for the reachability problem. *Fundamentia Informatica* 46:1–17.

Esparza, J.; Römer, S.; and Vogler, W. 2002. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design* 20(3):285–310.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An approach to temporal planning and scheduling in domains with predicatable exogenous events. *Journal of AI Research* 25:187–231.

Haslum, P. 2006. Improving heuristics through relaxed search – an analysis of TP4 and HSP$^*_a$ in the 2004 planning competition. *Journal of AI Research* 25.

Helmert, M. 2006. *Solving Planning Tasks in Theory and Practice*. Ph.D. Dissertation, Institut für Informatik, Albert-Ludwigs Universitaẗ Freiburg.

Hickmott, S.; Rintanen, J.; Thiébaux, S.; and White, L. 2007. Planning via Petri net unfolding. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, 1904–1911.

Hoffmann, J. 2003. The metric-FF planning system: Translating ignoring delete lists to numerical state variables. *Journal of Artificial Intelligence Research* 20.

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. SATPLAN: Planning as satisfiability. In *5th International Planning Competition Booklet*. Available at `http://zeus.ing.unibs.it/ipc-5/`.

Khomenko, V., and Koutny, M. 2003. 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)*, 458–472.

McMillan, K. L. 1992. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proceedings of the Fourth International Workshop on Computer-Aided Verification (CAV-92)*, 164–177.

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, 326–333.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence* 170(3):298–335.

Vogler, W.; Semenov, A. L.; and Yakovlev, A. 1998. Unfolding and finite prefix for nets with read arcs. In *Proceedings of the ninth International Conference on Concurrency Theory (CONCUR-98)*, 501–516.