Directed Unfolding of Petri Nets

Sarah Hickmott, Sylvie Thiebaux, Blai Bonet, P@trik Haslum

NICTA/UofA/USB

UFO'07

Australian Government
Department of Communications, Information Technology and the Arts
Australian Research Council

NICTA Members
ANU THE AUSTRALIAN NATIONAL UNIVERSITY
UNSW
Department of State and Regional Development
BusinessACT ACT GOVERNMENT

Victoria The Place To Be
THE UNIVERSITY OF MELBOURNE
The University of Sydney
Queensland Government
Griffith UNIVERSITY
QUT Queensland University of Technology
THE UNIVERSITY OF QUEENSLAND AUSTRALIA
NICTA Partners

Motivation

- **Reachability Analysis**:
  - Interested only in if a particular transition t_G (or state/marking M_G) is reachable.
  - Reachability algorithm should make use of this information!
- **Unfolding**:
  - Finds all possible runs of the net, hence all reachable markings.
  - On-the-Fly Reachability Analysis: Stops as soon as t_G is added to the unfolding.
- **Directed Unfolding**:
  - Guide search towards the sought transition (marking).
  - Using heuristic functions (extracted automatically from the net).

Outline

I. An Introduction to Heuristic State-Space Search

II. Unfolding & Directed Unfolding.

III. Realisation & Experimental Results

Part I: Heuristic State-Space Search

A State Space

A graph $S = (S_S, T_S)$, where

- S_S is the set of states,
- T_S is the transition relation.
- Edges in T_S labeled with non-negative costs, $cost(s, s')$.
- A distinguished initial state, $s_I \in S_S$.
- A set of goal states, $G \subset S_S$.

The Problem

- Find a path in S from s_I to any $s' \in G$, minimising the sum of edge costs along the path.
- S is only given in some implicit ("factored", "structured") – exponentially compact – representation.
- Representation imposes structure on S.

Examples of Representations

1-Safe Petri Nets

- States: Assignments of $0/1$ tokens to each place.
- Transitions as defined by the net.
- *etc.*

Network of Synchronised Automata

- States: Cross-product of component automata states.
- Transitions as defined by component automata and synchronisations.
- *etc.*

Examples of Representations

Propositional Planning (STRIPS)

- States: Assignments of truth values to a set of proposition symbols, $p_1, \ldots, p_n$.
- Transitions defined by actions: Each action a has a set of preconditions ($pre(a)$), sets of positive ($add(a)$) and negative ($del(a)$) effects, and a constant cost.
- a is applicable in s if each $p \in pre(a)$ true in s. a applied in s leads to a state s' where:
  - p is true if $p \in add(a)$
  - p is false if $p \in del(a)$
  - p keeps the truth value it had in s otherwise.
- Goal states are defined by a subset of propositions required to be true.

Blind and Directed Search

Generic Graph Search Algorithm

```
(1)   place s_I on queue;
(2)   while (queue not empty)
(3)     let s = first node in queue;
(4)     if (termination test(s))
          we're done;
(5)     if (s already reached)
(6)       if (new path to s is cheaper)
(7)         update graph and queue;
        else
(8)       insert s in graph;
(9)       for each s' such that S_T(s,s')
(10)        place s' on queue;
```

Blind and Directed Search

- Builds an explicit representation of a reachable fraction of S.
- Different algorithms characterised by queue ordering and termination test.

- $g(s)$: Cost of the (cheapest known) path from s_I to s.
- $h(s)$: Estimated cost of cheapest path from s to some $s' \in G$ (heuristic).
- $f(s) = g(s) + h(s)$: Estimated cost of (cheapest) path from s_I to any $s' \in G$, through s.
- $h^*(s)$: Actual cost of cheapest path from s to some $s' \in G$.
- f^*: Cost of optimal path from s_I to some $s' \ inG$.

Blind and Directed Search

Blind (Uniform Cost) Search

- Queue ordered by increasing path cost ($g(s)$).
- Stops when first $s \in G$ dequeued: path cost to s is minimal.
- When all transition costs equal to 1: Breadth-First Search.

A*

- Queue ordered by increasing $f(s) = g(s) + h(s)$.
- Stops when dequeued $s \in G$ or $f(s) = \infty$.
- Completeness & optimality depend on properties of h.
- When $h \equiv 0$: blind search.

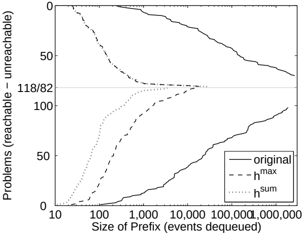
Properties of the Heuristic

- h is non-negative and $h(s) = 0$ when $s \in G$.
- h is safely pruning iff $h(s) = \infty$ implies $h^*(s) = \infty$, $\forall s$.
- h is admissible iff $h(s) \leqslant h^*(s)$, $\forall s$.
- h is monotone iff $h(s) \leqslant cost(s, s') + h(s')$, $\forall s, \forall s' : S_T(s, s')$.
- Monotonicity implies admissibility which implies pruning safety.

- If h is safely pruning, then A* is complete.
- If h is admissible, then the path found by A* has minimal cost.
- If h is monotone, then A* finds a cheapest path first to any state (lines (6) – (7) never invoked).

Why Non-Admissible Heuristics?

- *h* is informative if it directs the search quickly to a goal state!
- Admissible heuristic estimates need to be conservative (may not over-estimate) – therefore often less discriminating.
- When the goal is unreachable, what matters is the pruning power, *i.e.*, the heuristics ability to detect dead end states.
- In practice, any reasonable heuristic is safely pruning.



Part II: Petri Net Unfolding

The ERV Unfolding Algorithm (Esparza *et al.*, 2002)

- Constructs an explicit representation of all partially ordered runs of a Petri net (known as the finite prefix).
- Parameterised by an order on configurations, used to:
  - order the queue (*i.e.*, determine order in which events are inserted into the prefix), and
  - define cut-off events (discontinued branches).
- This order is required to be adequate:
  (a) $<$ is well-founded;
  (b) $C \subset C'$ implies $C < C'$;
  (c) $C < C'$ and $mark(C) = mark(C')$ implies $C + E < C' + E$, for any finite extension E.
- Normal order, $C < C'$ iff $|C| < |C'|$, equates to blind search.

Directed Unfolding

Let $f(C) = g(C) + h(mark(C))$, where

- $g(C)$ is the cost of C (standard: $g(C) = |C|$), and
- $h(M)$ is the estimated "distance" from M to the target transition/marking.

Define $C \prec_f C'$ iff

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

- h is a function of the marking: if $mark(C) = mark(C')$ then $f(C) < f(C')$ iff $g(C) < g(C')$.
- If h is monotone, then $\prec_f$ is adequate.

What's in the Paper...

Semi-Adequate Order

Replace condition (b) by

(b') in any sufficiently long chain $C_1 \subset C_2 \subset \ldots \subset C_m$, there exist $1 \leq i < j \leq m$ such that $C_i < C_j$.

- $\prec_f$ is semi-adequate for any (sane) heuristic function.

Observation #1

The finiteness proof by Esparza *et al.* works as well with property (b') as with (b).

Observation #2

The completeness proof by Esparza *et al.* does not depend on property (b) at all.

Completeness

- Let $\prec_{cut}$ be any adequate order.
- **Search scheme:** event e is terminal iff
  - e is the "target event", or
  - e leads to the same marking as some e' that we have already seen, and $[e'] \prec_{cut} [e]$.

Theorem

This scheme is complete with any search strategy (queue order).

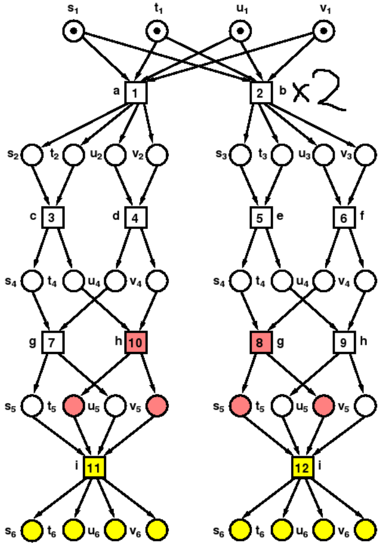
Proof (idea)

The prefix built in this way contains everything that the prefix built using $\prec_{cut}$ as the strategy would (modulo early termination).

Unfolding with Non-Monotone Heuristics

- $\prec_f \cap \{(C, C') \mid \mathit{mark}(C) = \mathit{mark}(C')\}$ is adequate, for any heuristic (because h is a function of $\mathit{mark}(C)$ – $f(C) < f(C')$ iff $g(C) < g(C')$).
- ERV with $\prec_f$, using any heuristic, is complete.
- If heuristic h is "wrong", the prefix may become (much) larger.
- But, in practice, it is (much) smaller, because
  - we stop when we reach the target transition, and
  - if h is safely pruning, we can stop when the f-value of the next (cheapest) event on the queue is ∞.

The Example



- Assume $\prec_{cut}$ is the standard order (cardinality).
- "h (10)" is not terminal!
- "g (8)" is a "junk event" (would not have been added to the prefix if exploration followed $\prec_{cut}$).

NATIONAL ICT AUSTRALIA LIMITED

Part III: Implementation & Results

- Implemented in MOLE (implements ERV algorithm).
- Three different heuristics:
  - h^{max} – monotone.
  - h^{sum} – non-admissible.
  - h^{FF} – non-admissible.

  All three are safely pruning.
- No additional tie-breaking.

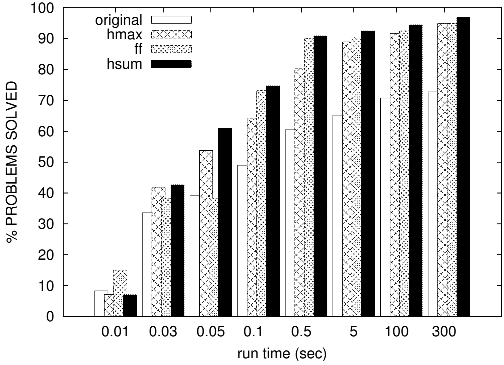
Heuristics: h^{max} and h^{sum}

- Heuristic value is cost (size) of solution to a relaxed problem.
- Relaxation: Assume independence between places.
- Conservative estimate (h^{max}): the cost of marking a set of places M equals the cost of marking the most expensive place $p \in M$.
- Non-admissible estimate (h^{sum}): the cost of marking a set of places M equals the sum of costs of marking each place $p \in M$.
- Formulate Bellman equation for relaxed problem, solve by dynamic programming ($O(|P|^2)$).

Heuristics: h^{FF}

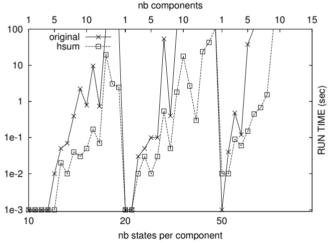
- Heuristic value is cost (size) of solution to a relaxed problem.
- Relaxation: Ignore conflicts, *i.e.*, treat two events consuming same token or writing to same place as non-conflicting.
- Two-phase solution:
  - Construct a "relaxed prefix" (**R**elaxed **P**lanning **G**raph) – polynomial size because each event fires at most once.
  - Extract solution from RPG, without search – time linear in RPG size.
- More informative than h^{max}, less over-estimating than h^{sum} – find shorter solutions.

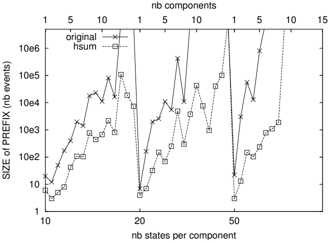
Results: DARTES



- Checked reachability of each of 253 transitions.
- Shortest solution lengths reach over 90 events – breadth-first search scales only to around 60.

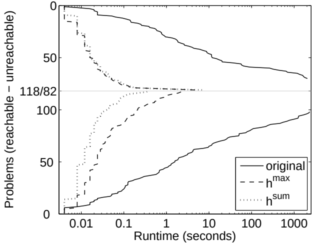
Results: Random Nets (1)

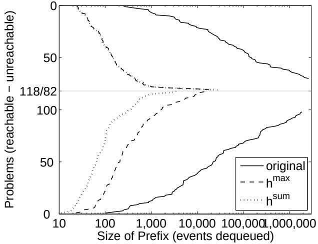




- Scaling from 10 / 30 to 750 / 4000 (places / transitions); goal marking always reachable.
- Shortest solutions of several hundred events; with h^{FF}, find solutions within a few events of optimal (where known).

Results: Random Nets (2)





- Smaller problems ($\sim$ 100 places / 200 transitions), some with unreachable goal markings.
- h^{max} (monotone) and h^{sum} (non-admissible) behave the same on unsolvable problems (same pruning power).

Conclusions

- Don't solve a harder problem than you have to:
  - If you only care about reachability of *one* marking, don't search for all (on-the-fly).
  - If you don't necessarily want an optimal solution, don't constrain search to find only optimal solutions.
- Unfolding is more "clever" than state-space search – but that's no excuse not to use a clever search strategy!
- Search in large, discrete spaces is a problem in many areas of computer science: Integrating techniques from different fields benefits everyone.
- There are many other heuristics to try out...

Addendum

- Planning via unfolding:
  - Model differences: "read-arcs" are frequent (essential) in most planning problems.
- Factored planning:
  - "Factored Planning: How, When and When Not" (Domshlak & Brafman, 2006).