Incremental Graphplan

David E. Smith and Daniel S. Weld

Technical Report UW–CSE–98-09–06
September 1998

Department of Computer Science and Engineering
University of Washington

Incremental Graphplan

David E. Smith
Nasa Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035 USA
de2smith@ptolemy.arc.nasa.gov

Daniel S. Weld
Dept. Computer Sci. & Engr
University of Washington
Seattle, WA 98195–2350 USA
weld@cs.washington.edu

January 19, 1999

1 Introduction

One can avoid duplicated work during plan expansion (or regression focussing) by exploiting the following observations concerning monotonicity in the planning graph.

- *Propositions are monontonicaly increasing:* if proposition P is present at level i it will appear at level $i+2$ and in all *subsequent* proposition levels. Proof sketch: if P appears at i then a nop will preserve it to the next proposition level.
- *Actions are monontonicaly increasing:* if action A is present at level i it will appear at level $i+2$ and in all *subsequent* action levels. Proof sketch: if an action's preconditions appear at level $i-1$ then nops will preserve them, and no new mutexes will appear.
- *Mutexes are monotonically decreasing:* if mutex M between actions A and B is present at level i then M is present at all *previous* action levels in which both A and B appear. The same is true of mutexes between propositions. Proof sketch: if A and B appear at both level i and $i-2$ and are mutex at level i then by definition this mutex must be due to inconsistent effects, interference, or competing needs. If the mutex is due to the first two reasons then the mutex will occur at *every* level containing A and B. However, if the mutex is due to competing needs then there are preconditions P and Q of A and B respectively such that P is mutex with Q at level $i-1$. This propositional mutex can only result from the fact that all level $i-3$ actions supporting P and Q are pairwise mutex, so an inductive argument (combined with action monotonicity) completes the proof.

- *Nogoods are monotonicaly decreasing:* If subgoals P, Q, and R are unachievable at level i then they are unachievable at all previous proposition levels. Proof sketch: if they were achievable at level $i-2$ then adding a level of nops would produce a plan at level i.

These observation suggest that one can dispense with a multi-level planning graph altogether. Instead, all one needs is a bipartite graph with action and proposition nodes. Arcs from propositions to actions denote the precondition relation and arcs from actions to propositions encode effects. Action, proposition, mutex, and nogood structures are all annotated with an integer label field; for proposition and action nodes this integer denotes the *first* planning graph level at which the proposition (or action) appears. For mutex or nogood nodes, the label marks the *last* level at which the relation holds. By adding an additional set of labels one may interleave forward and backward expansion of the planning graph. Using this scheme, the space costs of the expansion phase are vastly decreased.

2 Incremental Update

Using the representation outlined above, it is possible to update the graph in an incremental fashion. More precisely, we can keep track of what has changed in the graph, and only examine those propositions, actions and mutex relationships that can be affected by the changes. In particular:

- Adding a proposition to the proposition set can result in actions added to the action set
- Adding an action to the action set can cause propositions (effects) to be added to the proposition set, and/or can cause mutex relationships among propositions (effects) to terminate.
- Terminating a mutex relationship between propositions can cause actions to be added to the action set, and/or cause mutex relationships between actions to terminate.
- Terminating a mutex relationship between actions can cause mutex relationships between propositions (effects of the actions) to terminate.

This is illustrated in the causation diagram of figure 1.

Using this incremental approach, the time required in the expansion phase is reduced in proportion to the reduction in space. The bookeeping for this incremental update is surprisingly tricky.

3 Definitions and Notation

There are really two different types of mutex relationships, those that can go away, and those that are permanent. We refer to these as eternal and circum-

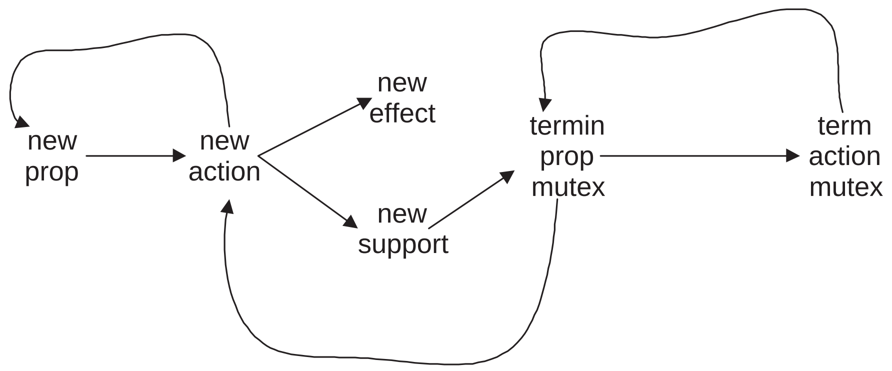

Figure 1: Causation Diagram

stantial mutex relationships. (This distinction turns out to be useful in the algorithms below, because it saves a certain amount of checking.)

Definition: Two propositions are eternally mutex iff the propositions are contradictory (negations of each other). Two propositions are circumstantially mutex iff they are not inherently mutex and all actions giving rise to them are mutex.

Definition: Two actions are eternally mutex iff either

- one action clobbers the other's preconditions or effects, or
- the actions have eternally mutex preconditions

Two actions are circumstantially mutex iff they are not inherently mutex and their preconditions are circumstantially mutex.

The following notation is useful.

1. Assume all actions have the same duration (two time units) This is a bit strange, but keeps times consistent with standard graphplan level numbering
2. Time is numbered with propositions at even times, actions at odd times
3. If prop p is labeled 2. written `p@2`, it first appears at time 2. Sometimes we say `p[2`
4. If mutex m is labeled i writtem `m@i`, it's present at i but no higher. Equivalently `m)i+2`

4 Expansion Algorithm

```
Let t := 0
Let newprops := the set of literals that are initially true
```

```
For each p in newprops do
    Add p@t to the graph
Let term-p-mutex-pairs := {}   ; note this is a list of pairs, not of mutexes
Loop
    If all goals present and nonmutex do solution extraction else
    t := t+1
    Let A = the set of newly executable actions
        (i.e. those not in the graph but whose preconds are present nonmutex)
        ; Consider nop actions explicitly
        ; An optimization: only need consider actions w/ a precond in newprops
            ; Or two preconds related by a mutex in term-p-mutex
    Let newprops, newsupp := {}
    Forall a in A do
        Add a@t to the graph
        For each literal p in the effect of a
            If p is already in the graph
                Then push p on newsupp; link a to p
                Else push p on newprops; add p@t+1 to graph; link a to p;
        ; Now add all eternal mutexes between a and other actions in the graph:
        ; 1) the actions can have opposite effects,
        ; 2) one action can clobber the other's preconditions, or
        ; 3) the actions' preconditions can be eternally mutex
        ; And add circumstantial mutexes between a and other actions present
        For each effect e of a:
            If ~e exists then add an eternal mutex
                between a and all producers or consumers of ~e
            For each precondition p of a
                If ~p exists then add an eternal mutex
                    between a and all producers of ~p
                For each q mutex with p
                    If the mutex is eternal then add an eternal mutex
                        between a and all consumers of q
                    Else add a circumstantial mutex (with nil label)
                        between a and all consumers of q ; if not already exist
    For each pair <p,q> in term-p-mutex-pairs do
        For each action a with p as precond
            For each nil-labeled circumstantial mutex m2 between a and b
                If b has q as precond
                Then pairwise check all preconds of a,b
                    If a,b no longer mutex,
                    Then label m2 with t-2, and  ; or label "m)t"
                        add all effects of a (or of b) to newsupp
        ; Note: this could be optimized by sorting term-p-mutex-pairs on <p,>
        ; And then coalesing <p,q> with adjacent <p, r> to get <p, {q,r}>
        ; And checking both q and r and ... in the line "If b has q as precond"
        ; Note line labeled "***" ensures that eliminated action mutexes
        ; cause subsequent prop mutexes to disappear when appropriate.
        ; is there a more efficient way?
    Let t := t+1
    Let term-p-mutex-pairs = {}
    For each p in newprops
        Add all eternal mutexes between p and old or previously-handled props
        ;; better to add the eternal mutexes earlier (with actions)?
        Add circumstantial mutexes between p and ... (with nil label)
            ; Note: we know all supporting actions by now, so this is fine
    For each p in newsupp
        For each circ mutex m (with nil label) between p and some other prop q
            ; Note avoid trying pq AND qp by restricting to canonical order p<q
        Check m's mutex conditions (are all supporting actions pairwise mutex
            either eternally or circumstantially with label of t-1 or nil)
            If not, label m with t-2
            Add <p,q> to term-p-mutex-pairs
```

5 Regression Focussing

Using a slight modification to the data-structures described above, one may do completeness-preserving regression focussing in an incremental fashion as well. The basic idea is to associate another set of labels with actions and propositions which records how many time steps *before* the goal these nodes become relevant. For goal propositions, these labels would be zero. Action that produced goal conjuncts would have a label of negative one, and preconditions of these actions would have a label of negative two. One can incrementally update these labels as well using whatever control strategy one wishes. For example, one could alternate expanding the graph (in-place) forward with a regression phase. Backward-chaining solution extraction need only be enabled when the frontiers from these two phases meet.