Temporal Planning with Mutual Exclusion Reasoning

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

Daniel S. Weld
Department of Computer Science & Engineering
University of Washington, Box 352350
Seattle, WA 98195-2350 USA
weld@cs.Washington.edu

Abstract

Many planning domains require a richer notion of time in which actions can overlap and have different durations. The key to fast performance in classical planners *(e.g.,* Graphplan, IPP, and Blackbox) has been the use of a disjunctive representation with powerful mutual exclusion reasoning. This paper presents TGP, a new algorithm for temporal planning. TGP operates by incrementally expanding a compact planning graph representation that handles actions of differing duration. The key to TGP performance is tight mutual exclusion reasoning which is based on an expressive language for bounding mutexes and includes mutexes between actions and propositions. Our experiments demonstrate that mutual exclusion reasoning remains valuable in a rich temporal setting.

1 Introduction

For many real world planning domains, the classical STRIPS model of action is inadequate — actions can be simultaneous, can have different durations, and can require metric resources. These characteristics are particularly prevelant in many NASA planning applications. For example, both spacecraft (such as DS1) and planetary rovers (such as Sojourner) use heaters to warm up various components, and these warming actions may span several other actions or experiments. Likewise, data compression and telemetry may overlap with other actions, and these actions may have wildly different durations (from milliseconds to hours).

While previous work on temporal planning has yielded some success [Vere, 1983; Pelavin &. Allen, 1987; Penberthy & Weld, 1994; Muscettola, 1994], past systems either scaled poorly or required humans to set up elaborate temporal constraint networks and specify guidance heuristics. The use of reachability analysis and mutual exclusion reasoning in Graphplan [Blum & Furst, 1995] and descendants such as IPP [Koehler *et al.*, 1997] has yielded spectacular speedup in classical planning, so it is natural to wonder if similar reasoning is extensible to the problem of temporal planning. This paper demonstrates that this extension is indeed possible in the generalized Graphplan context. In particular, we:

- Generalize the planning graph representation to deal with arbitrary time instead of graph levels. To accomplish this, we change to a much more compact cyclic graph, where actions and propositions appear only once in the graph annotated by their earliest possible start times.
- Extend mutual exclusion reasoning to work for actions that can have different durations and can overlap in arbitrary ways. This requires 1) a more general notion of *conditional mutex involving time bounds, and 2) mutex relationships between actions and propositions.*
- *Describe the Temporal Graphplan (*TGP*) algorithm, which operates incrementally on the generalized planning graph introduced above, and employs extended mutual exclusion reasoning on that graph.*
- *Present empirical evidence that 1) these generalizations do not significantly degrade performance, and 2) mutual exclusion remains valuable (and perhaps vital) in a richer temporal setting.*

2 Graphplan Review

We briefly summarize the Graphplan algorithm [Blum & Furst, 1997], because it forms the basis for TGP. Graphplan solves STRIPS planning problems in a deterministic, fully specified world. Both the preconditions and effects of its action schemata are conjunctions of literals (i.e., denoting the add and delete lists). Graphplan alternates between two phases: *graph expansion* and *solution extraction.* The graph expansion phase extends a *planning graph* until it has achieved a necessary (but in-

*We thank Corin Anderon, Keith Golden, Zack Ives, Ari K. Jonsson, Rao Kambhampati, Pandu Nayak and the anonymous reviewers for helpful comments and discussion. This research was funded in part by Office of Naval Research Grant N00014-98-1-0147, and by National Science Foundation Grants IRI-9303461 and IIS-9872128.

sufficient) condition for plan existence. The solution extraction phase then performs a backward-chaining search for an actual solution; if no solution is found, the cycle repeats.

The planning graph contains alternating levels of *proposition nodes* (corresponding to ground literals) and *action nodes.* The zeroth level consists solely of the propositions that are true in the initial state of the planning problem. Nodes in an action level correspond to action instances; there is one such node for each action instance whose preconditions are present (and are mutually consistent) at the previous proposition level. Directed edges connect proposition nodes to subsequent action nodes whose preconditions reference those propositions. Similarly, directed edges connect action nodes to subsequent propositions made true by the action's effects. Persistence actions function like frame axioms: each proposition at a level is linked to its persistence action at the next level, and the action connects to the same proposition at the next level. Graphplan defines a binary mutual exclusion relation ("mutex") between nodes in the same level. For example, two action instances are mutex if one action deletes a precondition or effect of another or the actions have preconditions that are mutually exclusive at the previous level. Two propositions are mutex if all ways of achieving the propositions *(i.e.,* actions at the previous level) are pairwise mutex.

Suppose that Graphplan has extended the planning graph to a level in which all goal propositions are present and none are pairwise mutex. Graphplan now searches for a solution plan by considering each goal conjunct in turn. For each such proposition, Graphplan chooses (backtrack point) an action *a* at the previous level that achieves the goal. If *a* is consistent (nonmutex) with all actions that have been chosen so far at this level, then Graphplan proceeds to the next goal, otherwise if no such choice is available Graphplan backtracks. After Graphplan has found a consistent set of actions it recursively tries to find a plan for the actions' preconditions at the previous proposition level. The base case for the recursion is level zero if the propositions are present in the initial conditions, then Graphplan has found a solution. Otherwise, if backtracking fails, then Graphplan extends the planning graph with an additional action and proposition level and tries again.

3 The Temporal Planning Graph

When talking about temporal actions, it is important to specify a clear semantics. TGP adopts a simple extension of the STRIPS action language that allows each action to have a nonnegative start time, s, and a positive, real-valued duration, *d.* We adopt a conservative model of action in which: 1) all preconditions must hold at the start, .$, of the action, 2) preconditions not affected by the action itself must hold throughout execution, [s, s + d], and 3) effects are undefined during execution and only guaranteed to hold at the final time point s + *d.* This means that two actions cannot overlap in *any* way if an effect or precondition of one is the negation of an effect or precondition of the other.

One can achieve a more flexible representation of the planning graph (and at the same time avoid duplicated work during plan expansion) by exploiting the following observations:

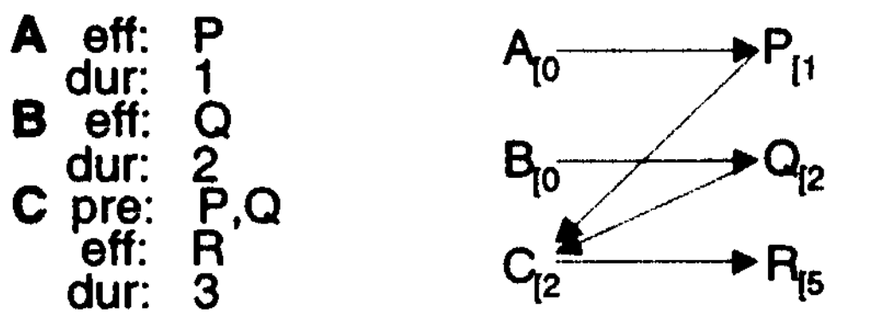
- *Propositions and actions are rnonotonically increasing:* if proposition P (or action *A)* is present at one level it will appear at all *subsequent* proposition (action) levels.
- *Mutexes are rnonotonically decreasing:* if mutex *M* between propositions *P* and *Q* is present at one level then *M* is present at all *previous* proposition levels in which both *P* and *Q* appear. Mutexes between action instances behave similarly.
- *Nogoods are rnonotonically decreasing:* If subgoals P, *Q,* and *R* are unachievable at a level then they are unachievable at all previous proposition levels.

These observations show that one can dispense with a multi-level planning graph altogether. Instead, all one needs is a 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 a numeric label field; for proposition and action nodes this number 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.

Note that storing a single node in the graph per action does not limit the planner to a single *instance* of the action in a plan. Solution extraction will search through the graph, adding action instances into the plan. Since this backward-chaining search may traverse cycles, multiple instances of an action may be added into a plan. Indeed, this compact encoding scheme has three advantages:

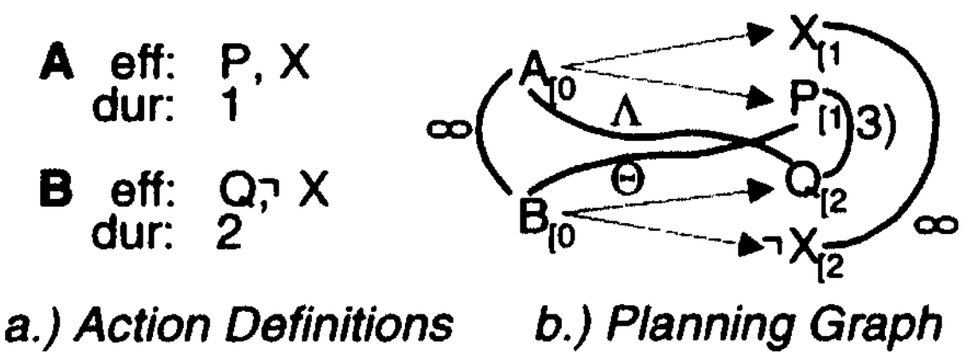
1. The space costs of the expansion phase are vastly decreased, because information is not duplicated between levels.
2. The speed of the expansion phase is increased, because it is possible to update the graph in an incremental fashion. We elaborate on this point in Section 5.
3. Most important, when using this representation, there is no longer any need to have actions take unit time. Instead, labels can be real numbers denoting start times instead of integers marking a planning-graph level. While this idea is conceptually simple, it hides a surprising number of subtleties, which we elaborate upon in subsequent sections.

As an example, consider the simple domain shown in Figure 1. Actions *A* and *B* have no preconditions, and produce *P* and *Q* respectively. Since action *C* requires both P and Q, it can't start executing until time 2 and so the earliest *R* can be produced is time 5.



a.) Action Definitions *b.) Planning Graph*

Figure 1: The planning graph for a simple domain of three actions. Arcs encode precondition and effect relations. Node subscripts, which start with an open bracket, indicate the earliest time that the action (or proposition) can be executed (or achieved).



a.) Action Definitions ***b.) Planning Graph***

a.) Action Definitions *b.) Planning Graph*

Figure 2: A simple domain that illustrates the need for action/proposition mutexes. Bold lines denote mutexes (regardless of type). Labels on mutexes denote the conditions when the mutex holds. The oo signifies that *X* and -X" are eternally mutex. *A* is cmutex with *Q* when A and *B* is cmutex with *Q* when 0, where A = *[A < [Q* and 6 = *[B < [P.*

4 Generalized Mutex Reasoning

This section generalizes the Graphplan mutex rules in two ways: 1) by introducing action /proposition mutexes (in addition to the original mutexes between pairs of actions or between pairs of propositions), and 2) by distinguishing between mutex relations that are eternally present and those that are conditional and may expire as the graph is expanded further in time. We motivate these enhancements with the example of Figure 2. Because actions *A* and *B* produce *X* and -X respectively, they can *never* be executed at the same time. (We formalize this notion below as an *eternal mutex,* and depict it as a mutex with oo label in the planning graph.) The only way to achieve both *P* and *Q* is to execute *A* and *B* in series (the order doesn't matter), so the mutex between *P* and *Q* *should* end at time 3. But standard Graphplan mutex propagation is insufficient for deducing this fact.

The problem stems from the fact that actions have differing durations. While the original Graphplan approach works when proposition and action levels alternate in a regular fashion, actions with varying durations break this symmetry. For example, even if *P* is made true early and persists to time 2, the action making *Q* true may span backwards far enough to overlap the source of *P*\ and, indeed, if *B* overlaps *A* there is a conflict.

We repair this reasoning limitation by introducing the notion of an action/proposition mutex. Note that in the previous example, it is impossible to have proposition *P* true and have an instance of action *B* under execution at time 2 if *B* *started* execution before *P* became true. Intuitively, action/proposition mutexes help deduce more inconsistencies because they better connect action/action mutexes to proposition/proposition mutexes in cases where action executions overlap. We now make these notions precise.

We partition all mutex relations (action/action, proposition/proposition, and action/proposition) into eternal and conditional types for efficiency purposes. Intuitively, an eternal mutex unconditionally persists for all time, while a conditional mutex might not always hold. Formally, we say that

Def 1. Propositions *P* and *Q* are *eternally mutex (emutex)* iff *P* is the negation of *Q,*

Def 2. Action *A* is *emutex* with proposition *P* if -»P is a precondition or effect of *A,* or if *P* is an effect of *A.*

Def 3. Action *A* is *emutex* with action *B* iff at least one of the following holds: 1) *A* or *B* deletes the preconditions or effects of the other, or 2) *A* and *B* have emutex preconditions.

In contrast to emutex, a conditional mutex may be transitory, applying early on but expiring later due to additional support for a proposition. Typically, the conditions governing when a emutex applies are inequalities referring to:

- The duration of an action, which we write: \A\
- The time when an action (or proposition) first appears in the planning graph: [A
- The earliest possible end time of an action: A]
- The time when an action *instance* starts executing, $\lfloor A$, or a proposition *actually* becomes true: $\lfloor P$
- The completion time of an action *instance:* $A\rceil$

Note that $A\rceil = [A + |A|$, that $A\rceil = \lfloor A + |A|$, and that $[A \leq \lfloor A$. We now state three definitions, pertaining to proposition/proposition, action/proposition, and action/action pairs.

Loosely speaking, propositions *P* and *Q* are emutex when *P* is emutex with all of the actions supporting *Q* and also vice versa. The inqualities in the formalism below ensure that an action is counted as support only when it ends before the proposition starts.

Def 4. Let *P* and *Q* be two propositions. For each *Ai* supporting *P* let Φ_i be the condition under which *Ai* is mutex with *Q* (true if eternally mutex, false if no mutex, and Υ if A_i is emutex with *Q* when Υ). For each B_j supporting *Q* let Ψ_j be the condition under which B_j is mutex with P.

Let $\Phi = \bigwedge_i(\Phi_i \vee (A_i\rceil > \lfloor P))\wedge\bigwedge_j(\Psi_j \vee (B_j\rceil > \lfloor Q))$. If Φ is satisfiable, then *propositions P and Q are conditionally mutex (emutex) when* Φ

Intuitively action *A* is emutex with proposition *P* when *P* is emutex with *any* precondition of *A* *or* when *A* is emutex with *all* of the actions supporting *P.*

Def 5. Let *A* be an action and P be a proposition. For each precondition Q_i of A, let Φ_i be the condition under which P is mutex with Q_i (true if emutex, ...). For each action *Bj* possibly supporting P, let Ψ_j be the condition under which *A* is mutex with *Bj.*

Let $\Phi = (\bigvee_i \Phi_i) \vee \left(\bigwedge_j (\Psi_j \vee (B_j] > \lfloor P))\right) \wedge (\lfloor A < \lceil P)$. If Φ is satisfiable, then *action A and proposition P are emutex when* Φ

Loosely speaking, actions *A* and *B* are emutex when *A* is emutex with *any* precondition of *B or* vice versa.

Def 6. Let A and B be two actions which are not emutex. For each precondition P_i of *B,* let Φ_i be the condition under which *A* is mutex with P,. For each precondition Q_j of *A,* let Ψ_j be the condition under which *B* is mutex with *Qj.* Let $\Phi = \bigvee_i \Phi_i \vee \bigvee_j \Psi_j$. If Φ is satisfiable, then *actions A and B are emutex when* Φ.

To see how these rules work, consider the example of Figure 2. Def 1 shows that *X* is emutex with ~^*X.* Def 5 further concludes that *A* is emutex with *Q* when $\lfloor A < \lceil Q$, *i.e.* $\lfloor A < 2$; also that *B* is emutex with P when $\lfloor B < \lceil P$, *i.e.* $\lfloor B < 1$. Intuitively, this makes sense — if *A* starts before 2, then it must overlap support for *Q (i.e.,* action *B),* but *A* and *B* are emutex. Finally, Def 4 shows that propositions P and *Q* are emutex when $(\lfloor A < 2) \wedge (\lfloor B < 1)$. Adding the action durations to both sides of each inequality yields the following P/Q condition: $(A] < 3) \wedge (B] < 3)$. Thus we conclude that P and *Q* are emutex when $(\lfloor P < 3) \wedge (\lfloor Q < 3)$, and this simple, symmetric condition is equivalent to a standard Graphplan proposition mutex that expires at time 3.

In bigger examples, the situation gets more complicated and asymmetric. Being able to quickly manipulate and simplify the emutex conditions is a necessary ability for doing mutex reasoning with actions of varying duration. In Section 7, we describe a canonical form for these asymmetric conditions and explain how to quickly manipulate the conditions. In section 8, we present empirical evidence that mutex reasoning (while complex) yields important speedup.

5 Incremental Graph Expansion

Using the compact representation described above it is possible to update the planning graph in an incremental fashion. More precisely, the planner 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 node to the graph *(e.g.,* as the novel effect of a newly added action) can result in new actions (i.e., those with the proposition as precondition) being added.
- Adding an action to the graph can cause new propositions (the action's effects) to be added, and/or can provide additional support for existing propositions. This new support can cause an action/proposition emutex to terminate (by Def 5).

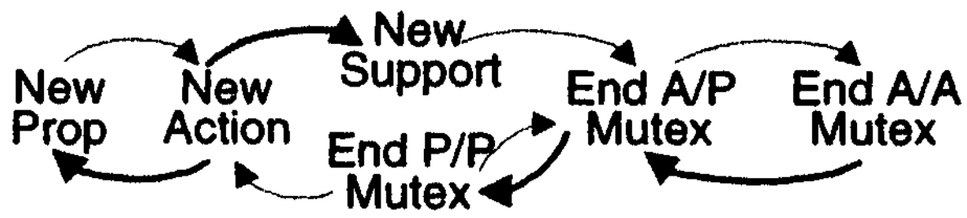


Figure 3: The TGP algorithm uses this *causation diagram* to guide its processing of events. Dark lines denote effects that occur later in time (i.e., after an action execution).

- Terminating a emutex between propositions P and *Q* can result in new actions *(e.g.,* actions with both P and *Q* as preconditions). In addition, it can cause an action/proposition emutex to end (by Def 5), *e.g.,* between P and a consumer, C, of *Q.*
- Terminating a emutex between action *A* and proposition P can cause a proposition/proposition emutex to terminate (by Def 4), *e.g.,* between P and an effect, *R, of A.* In addition, it can cause an action/action emutex to terminate (by Def 6), *e.g.,* between *A* and a consumer, *C,* of P.
- Terminating a emutex between actions *A* and *B* can cause an action/proposition emutex to end (by Def 5), *e.g.,* between *A* and an effect, *R,* of P.

These relationships are illustrated in the causation diagram of Figure 3. This diagram shows the structure of an incremental approach to graph expansion which provides speed gains proportional to the space reductions afforded by the compact representation.

Although the detailed bookkeeping is surprisingly complex, the basic TGP expansion algorithm is straightforward. Starting at time 0, it moves incrementally forward in time, progressively taking care of new propositions, new actions, new support for existing propositions and terminated mutexes. Persistence actions are *not* added explicitly. TGP keeps two main time-ordered priority queues, NewSupp and EndPPMutex. NewSupp contains triples *(A,P,t)* meaning proposition P has new support from action *A* at time *t,* and EndPPMutex contains pairs *(M,t)* meaning that *M* is a proposition/proposition emutex that ended at time *t.* For efficiency, TGP also keeps a temporary list: NewProps is the subset of propositions mentioned in NewSupp that are new *(i.e.,* have no prior support).

Given a temporal planning problem *(i.e.,* a set of initial conditions, list of conjunctive goals, and set of ground actions), TGP graph expansion follows a loop with the following steps:

1. Add new actions and their effects to the graph. Note: TGP need only consider actions with a precondition in NewProps, or with two preconditions whose emutex is in EndPPMutex. (At time zero, the initial conditions are added by a special instance of this step).
2. Add eternal and conditional mutex relationships for new actions; this includes both action/proposition and action /action mutexes.

3. Increment time to the next interesting entry in the RewSupp or EndPPMutex queues.
4. Recheck propositions with new support, possibly terminating *(i.e.,* tightening the bound on) action/proposition, action/action, and proposition/proposition cmutexes via a recursive algorithm that traverses the causation diagram.
5. Add action/proposition and proposition/proposition mutexes (both eternal and conditional) that involve new propositions.
6. If all goals are present in the graph, pair wise nonmutex, then call solution extraction. Otherwise (and if solution extraction fails) loop.

6 Solution Extraction

Once the planning graph has been extended to a time, *to,* when all goals are present and are pairwise nonmutex, TGP performs a backward chaining search for a working plan. This search is implemented using two main data structures: Agenda, and Plan. Agenda is a priority queue of $\langle P_i, t_i \rangle$ pairs, where P_i is a (sub)goal proposition and t_i, is the time by which the goal must be true. Agenda is initialized by enqueuing $\langle G_i, t_G \rangle$ for each top level goal G_i, and the queue is sorted in decreasing temporal order. The second structure, Plan, which is initialized empty, stores the plan under construction as a set of $\langle A_i, s_i \rangle$ pairs, where s_i, is the start time for action A_i. Persistence actions for a goal, denoted persist-G, are considered explicitly since they were not added during incremental graph expansion. The TGP solution extraction loop performs the following steps while Agenda is nonempty:

1. Dequeue *(G,t)* from Agenda.
2. If t= 0 and $G \neq$ initially true, then fail (backtrack). If $t > 0$ then let *S* equal the set of actions, $\{A_i\}$, such that each A_i has G as an effect and $A_i| \leq t$.
3. Choose *A* from S U {persist-G} such that *A* isn't mutex with any action in Plan. Add $\langle A, t - |A| \rangle$ to Plan, and for each precondition *P* of *A,* add $\langle P, t - |A| \rangle$ to Agenda. If no such *A* exists, backtrack. All such consistent *A's* must be considered for completeness.[1]

In essence, persistence actions are really just placeholders to ensure that TGP remembers to check all relevant action/proposition mutexes. Unfortunately, the presence of persistence actions adds redundancy to the space of plans, and this can lead to increased search. For a simple example of this, consider the domain of Figure 1 and suppose that the goal is to achieve both *P* and *Q.* The shortest plan involves executing actions *A* and *B* and requires two units of time. It should also be clear that *A* could start execution at any time in the interval [0,1]. But TGP should *not* consider all such times, for there is an uncountable number; indeed, the algorithm above restricts attention to start times which are an integral multiple of the greatest common divisor of the set of action durations, and this does not compromise completeness. For this example, the GCD restriction translates into starting *A* at time 0 or at time 1.

But TGP applies the following even stronger, completeness-preserving filter: all actions are executed as late as possible, unless this leads to a mutex inconsistency. Intuitively, one may think of this as defining a canonical form for plans by taking a legal plan and "tilting it" so that all actions "slide as far right" as they can go without breaking plan correctness. This completeness-preserving heuristic can be implemented by refusing to choose *A* = persist-G to support subgoal G (in step 3) unless all other choices are inconsistent.

7 Approximating Mutex Conditions

As we mentioned at the end of Section 4, simplifying the logical and inequality formulae that bound the applicability of conditional mutexes is a key component of temporal reasoning and a central aspect of the TGP algorithm. Since these formulae can get arbitrarily complex, we developed the *asymmetric* restricted form in an effort to keep the reasoning tractable. The asymmetric form limits the mutex condition to a simple conjunction of two inequalities, but allows for different bounds in each inequality. For example, *X* is mutex with *Y* when $(\lfloor X < t_x) \wedge (\lfloor Y < t_y)$. In contrast with the restrictive symmetric form, when one plugs this representation into Def 4, there is no loss of information. Unfortunate!y this is not the case for Def 5 and 6. For example,

Def 6a (Asymmetric). Let *A* and *B* be two actions which are not emutex. For each precondition *Pi* of *B,* let $\Phi_i = (\lfloor A < t_{A_i}) \wedge (\lfloor P_i < t_{P_i})$ be the condition under which *A* is mutex with P_i. For each precondition Q_j of *A,* let $\Psi_j = (\lfloor B < t_{B_j}) \wedge (\lfloor Q_j < t_{Q_j})$ be the condition under which *B* is mutex with Q_j. Let $\Phi = (\bigvee_i (\lfloor A < t_{A_i}) \wedge (\lfloor B < t_{P_i})) \vee \left(\bigvee_j (\lfloor B < t_{B_j}) \wedge (\lfloor A < t_{Q_j})\right)$

This condition does not simplify to the cannonical asymmetric form because of the A inside the V. There are several choices here for bounding approximations, and two possibilities are:

$$\Phi = (\lfloor A < \min(t_{A_i}, t_{Q_j})) \wedge (\lfloor B < \max(t_{B_j}, t_{P_i}))$$

and

$$\Phi = (\lfloor A < \max(t_{A_i}, t_{Q_j})) \wedge (\lfloor B < \min(t_{B_j}, t_{P_i}))$$

In practice we do something slightly more sophisticated by using the better of these max/min approximations on each successive pair of disjuncts. More precisely, the binary disjunction

$$((\lfloor A < x_1) \wedge (\lfloor B < y_1)) \vee ((\lfloor A < x_2) \wedge (\lfloor B < y_2))$$

[1] If *A* is the special "persist-G" action, let $|A|$ equal the greatest common divisor of the durations of the set of actions, and test for mutexes with proposition *G.*

becomes:

$$\begin{array}{ll} (\lfloor A < x_1) \wedge (\lfloor B < \max(y_1, y_2)) & \text{if } x_1 = x_2 \\ (\lfloor A < x_1) \wedge (\lfloor B < y_1) & \text{if } x_1 > x_2 \\ (\lfloor A < x_2) \wedge (\lfloor B < y_2) & \text{otherwise} \end{array}$$

We note that, this form gives an exact result except in two cases, 1) when $x_1 < x_2 \wedge y_1 > y_2$ and 2) $x_1 > x_2 \wedge y_1 < y_2$. In those two cases the result will be a rnin/max approximation.

Def 5a (Asymmetric). Let *A* be an action and *P* be a proposition. For each precondition Q_i of *A*, let $\Phi_i = (\lfloor P < t_{P_i}) \wedge (\lfloor Q_i < t_{Q_i})$ be the condition under which *P* is mutex with Q_i For each action B_j possibly supporting P, let $\Psi_j = (\lfloor A < t_{A_j}) \wedge (\lfloor B_j < t_{B_j})$ be the condition under which *A* is mutex with B_j Let $\Phi = (\bigvee_i (\lfloor P < t_{P_i}) \wedge (\lfloor Q_i < t_{Q_i})) \vee \left(\bigwedge_j (\lfloor A < t_{A_j}) \wedge (\lfloor B_j < t_{B_j}) \vee (B_j] > \lfloor P)\right) \wedge (\lfloor A < \lfloor P)$

If Φ is satisfiable, then *action A and proposition P are cmutex when* Φ

As before, this condition does not simplify to our cannonical form; there are several choices here for bounding approximations, and one rnin/max argument leads to the following:

$$\begin{aligned} \Phi \quad = \quad & (\lfloor P < \min(t_{P_i}, t_{B_j} + \lfloor B_j \rfloor)) \wedge \\ & (\lfloor A < \max(t_{Q_i}, \min(\lceil P, t_{A_j})))) \end{aligned}$$

Again, we improve on this equation by using the better of this and a symmetric max/min approximation on each successive pair of disjuncts.

8 Experimental Results

To date our implementation has been primarily used to verify the correctness and completeness of the cmutex rules; we have put little effort into code optimization. Direct comparison between TGP and other temporal planning systems is difficult for both availability and modularity reasons (HSTS [Muscettola, 1994], for example, is part of a larger embedded system and requires inputs which are radically different from classical representation). Nevertheless, we plan to do direct comparison in the immediate future.

In this section, we report on two experiments. First, we compare the performance of TGP with that of SGP [Weld, Anderson, & Smith, 1998] on plain STRIPS problems in order to see whether TGP's general, temporal framework comes at huge cost. Using a Power Mac G3/400 running Macintosh Common Lisp 4.2 in 68mb memory, we solved each problem ten times with SGP, with full TGP, and also using TGP with cmutex reasoning disabled. All runs were censored after 100 seconds, and we averaged across each set of ten runs to reach a single time for each problem/algorithm combination (Figure 4). TGP generally performs much better than SGP on the harder logistics problems, SGP wins on Med-bw2 and Big-bw2 which are dominated by solution extraction and appear sensitive to goal ordering decisions therein. We conjecture that SGP is faster at solution extraction either due to reduced overhead when checking mutexes or because of SGP's use of dynamic variable ordering [Bacchus & van Run, 1995].

Problem	TGP	No cmutex	SGP
Med-bw1	0.108	0.032	0.090
Big-bw1	0.529	>100.000	0.451
Med-bw2	1.034	46.019	0.418
Big-bw2	20.953	>100.000	4.535
Simple-block-stack	0.012	0.005	0.028
Simple-block1	0.023	0.006	0.042
Simple-block2	0.139	0.221	0.189
Simple-block3	0.723	8.219	0.560
Fix-strips1	0.024	0.011	0.329
Fix-strips2	0.026	1.443	0.546
Fix-strips3	0.026	2.602	0.546
Fix-strips4	0.054	60.079	0.840
Att-log0	0.014	0.010	3.352
Att-log1	0.018	0.015	7.332
Att-log2	0.026	0.115	11.692
Att-log3	1.516	3.453	>100.000
Att-log4	1.516	>100.000	>100.000
Log01	3.415	>100.000	>100.000
Log02	1.568	>100.000	>100.000
Strips-log-y-1	1.203	>100.000	>100.000
Strips-log-y-2	11.760	>100.000	>100.000
Strips-log-y-3	6.009	>100.000	>100.000

Figure 4: TGP with both cmutex and emutex reasoning beats emutex alone, and often beats SGP as well. Times are in seconds.

In our second experiment we considered TGP's performance on temporal planning problems, and again looked at the contribution of asymmetric cmutex reasoning. In the absence of a test suite of large temporal planning problems, we took 30 STRIPS problems (mostly from ATT logistics domains); for each STRIPS problem we created 10 temporal problems by randomly assigning actions a duration from a normal distribution of integers in the range [1, *x].* We then ran full TGP as well as TGP without cmutex reasoning on each of the resulting temporal planning problems, censoring after 100 seconds and averaging the results. We repeated this procedure for *x* = 2,4, and 8. Figure 5 displays the results clearly, cmutex reasoning provides a substantial gain in performance, especially for difficult problems.

We also note that TGP can handle relatively complex problems: *e.g.,* the solution to Log-4 is a 14-action plan; with emutex and cmutex reasoning combined, generation of the plan takes about 1.5 seconds on average.

9 Exogenous Events &: Time Windows

Thus far, our description of temporal planning has focussed on handling actions of extended duration, but several other aspects are equally challenging. A general temporal planner must also handle exogenous events (e.#., a solar eclipse or orbit perigee) and temporally constrained goals *[e.g.,* observations that must be performed during a time window). This section explains how the basic TGP algorithm can be extended with this functionality.

Suppose that as input the planning problem specified

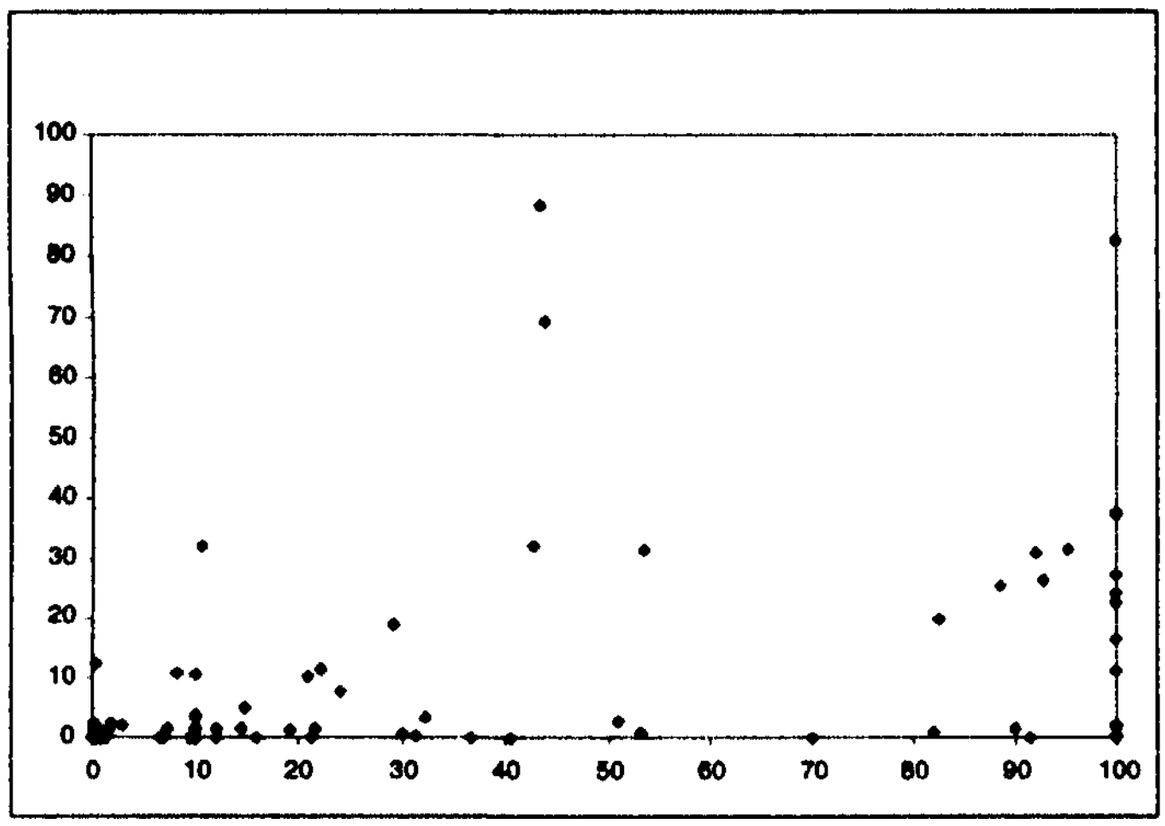


Figure 5: Cmutex reasoning provides substantial speedup on larger problems.

a set of (G, s, e) triples instead of a simple set of goals. We wish the planner to ensure that goal *G* is true at time *t* such that $s \leq t \leq e$. Handling this representation requires only minor changes to graph expansion and solution extraction. Graph expansion need never proceed past the maximum of the goal's endpoints, and TGP can claim failure without attempting solution extraction if some goal fails to enter the planning graph by its endpoint. Solution extraction is modified as follows:

1. Dequeue (G, s,e) from Agenda.
2. If $e = 0$ and $G \neq$ initially true, then fail (backtrack). If $e > 0$ then let *S* equal the set of actions, $\{A_i\}$, such that each $A\{$ has *G* as an effect and $A_i] \leq e$.
3. If $s < e$ then add persist-G to 5.
4. Choose *A* from *S* such that *A* isn't mutex with any action in Plan. Add $\langle A, e - |A| \rangle$ to Plan, and for each precondition *P* of *A,* add (P, 0,e — $|A|$) to Agenda. If no such *A* exists, backtrack. All such consistent *A'B* must be considered for completeness.

Note that this algorithm does not preclude *G* from being made true earlier than *s* and then persisting into the interval [s, e]. If one wishes to ensure that *G* is *made* true during that interval, one must post $\langle \neg G, 0, e - |A| \rangle$ as a goal in step 4.

There are (at least) two ways to handle exogenous events, and the first is simple. One can model exogenous events with a partial plan. Each event defines a special type of "action" with no preconditions but with the event's effects. Of course the agent has no choice about when these event "actions" are executed — these times are specified in the problem input. We denote the resulting set of (E,<) pairs with the variable Events. The following simplistic approach to graph expansion now suffices. During the normal process of expanding the planning graph, whenever the time is incremented to the starting time of an event *E* its effects are added into the graph. Similarly, solution extraction only requires minor modification: instead of initializing Plan to the empty set, it is initialized to equal Events. Normal mutex reasoning now ensures that the events will be dealt with correctly.

While correct, this approach is simplistic in its treatment of recurring events *(e.g.,* cyclic periods of blocked communications due to satelite orbits). Instead of storing a single time label on each proposition (and action and mutex) node in the planning graph, one should store a set of time intervals that dictate the times when the proposition is possibly achieveable. As we envision these extensions, the two level plan graph starts to look very much like a temporal CSP network, in which propositions, actions and mutexes "come and go," *i.e.* are active according to sets of allowed time windows. Unfortunately, efficient techniques like arc-consistency are not powerful enough to derive and propagate mutex relationships. For this, k-consistency is required. However, experience shows that general k-consistency reasoning is too unfocused to be practical. Mutex reasoning is a highly-focused form of k-consistency, and we believe it will prove quite valuable in more general temporal planning problems. Extending these techniques to a general temporal CSP is something we have just begun to investigate.

10 Related Work & Discussion

There is a long history of research on temporal planning, but few systems have seen wide use, presumably due to performance limitations. Deviser [Vere, 1983] is an early temporal planner which required a library of HTN schemata and numerous domain-specific heuristics. FORBIN [Dean, Firby, *k* Miller, 1988] combined HTN reduction and temporal projection to tackle a similar problem, but the system ran on only a few examples. IxTeT [Ghallab *k* Laruelle, 1994] is a more recent HTN-decomposition temporal planner. Allen *et al.* developed several elegant temporal planners based on temporal logic [Allen *k* Koomen, 1983; Pelavin *k* Allen, 1987; Allen, 1991], but none supported metric durations. The Zeno planner [Penberthy *k* Weld, 1994] used an incremental Simplex algorithm to support actions with metric durations and continuous change, but performance was lacking — TGP is orders of magnitude faster. HSTS [Muscettola, 1994] plans using a dynamic, temporal CSP. When the planner commits to an action, new nodes are added to the CSP corresponding to the action's start and end. Constraints are then added between the various time points in the CSP to specify action duration and to enforce precondition and effect constraints for the action. Since the result is a *simple* temporal network, arc-consistency is sufficient to determine overall consistency [Dechter, Meiri, *k* Pearl, 1991]. HSTS does not do any form of reachability analysis or mutual exclusion reasoning — it must commit to a particular action or event before it can do any reasoning about consistency. Although it does not handle actions of varying duration, STAN uses an independently developed representation akin to our compact planning graph [Long *k* Fox, 1999].

We now discuss a method for extending Graphplan to handle temporal actions without new mutex rules or our compact planning-graph representation. Instead, macro-expand each action in the domain into a number of atomic pieces, each the length of the GCD of the set of action durations, and each a regular STRIPS action. This compilation is a bit tricky since it needs to generate new propositions and add them as preconditions and effects of the different pieces in order to ensure that the pieces sequence properly and that two actions don't intercalate inappropriately. Unfortunately, this approach would vastly expand the size of the domain theory, if the ratio of the GCD of action durations is small relative to the longest action — which is inevitable if there is wide variation in action durations.

11 Conclusions

This paper makes several contributions:

- We describe TGP, a fast planner that handles temporally-extended actions.
- TGP incrementally generates a compact planning graph representation.
- The key to TGP'S performance is a novel form of reachability analysis for actions with varying duration. We distinguish conditional and eternal mutexes, and introduce action/proposition mutexes.
- We present experiments demonstrating the power of conditional mutex reasoning with an asymmetric condition representation.
- We explain how to extend TGP to handle exogenous events and goals that must be achieved during certain time windows.

We believe that the ideas introduced here can be extended to deal with a richer temporal language that allows: (1) for action preconditions which need not hold throughout execution *(i.e.,* "trigger" preconditions as well as "maintenance" preconditions), (2) for effects that become true during the action (instead of just at the end), and (3) temporary effects of actions *(e.g.,* inexhaustible resource usage). In section 9 we discuss methods for handling (4) exogenous events, and (5) time windows on goals and actions *(e.g.,* for modeling scientific experiments or astronomical observations), and we wish to experiment with the efficiency of these approaches. Yet, even without considering increased action expressiveness there are algorithm improvements to be made. Recall that during the backward chaining solution extraction phase, if a plan is not found then TGP initiates another search starting from a time which is a single GCD increment later. A more sophisticated approach would analyze the memoized nogoods and calculate the next time point when a nogood might vanish and start solution extraction there.

References

[Allen *k* Koomen, 1983] Allen, J., and Koomen, J. 1983. Planning using a temporal world model. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence,* 741-747.

[Allen, 199l] Alien, J. 1991. Planning as temporal reasoning. In *Proc. Conf. Knowledge Representation and Reasoning,* 3-14.

[Bacchus *k* van Run, 1995] Bacchus, F., and van Run, P. 1995. Dynamic variable ordering in csps. In *Proceedings of the 1995 conference on Principles and Practice of Constraint Programming,* 258-275.

[Blum *k* Furst, 1995] Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence,* 1636-1642. San Francisco, Calif.: Morgan Kaufmann.

[Blum *k* Furst, 1997] Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(l-2):281-300.

[Dean, Firby, *k* Miller, 1988] Dean, T., Firby, J., and Miller, D. 1988. Hierarchical planning involving deadlines, travel times, and resources. *Computational Intelligence*4(4):381-398.

[Dechter, Meiri, *k* Pearl, 1991] Dechter, R., Meiri, I., and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61-96.

[Ghallab *k* Laruelle, 1994] Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a Temporal Planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems,* 61-67. Menlo Park, Calif.: AAAI Press.

[Koehler *et a*/., 1997] Koehler, J., Nebel, B., Hoffmann, J., and Dimopoulos, Y. 1997. Extending planning graphs to an ADL subset. In *Proceedings of the Fourth European Conference on Planning,* 273-285. Berlin, Germany: Springer-Verlag.

[Long *k* Fox, 1999] Long, D., and Fox, M. 1999. The efficient implementation of the plan graph in STAN. *J. Artificial Intelligence Research* 10.

[Muscettola, 1994] Muscettola, N. 1994. HSTS: integrating planning and scheduling. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling.* Morgan Kaufmann.

[Pelavin *k* Allen, 1987] Pelavin, R., and Allen, J. 1987. A model for concurrent actions having temporal extent. In *Proceedings of the Sixth National Conference on Artificial Intelligence,* 246-250.

[Penberthy *k* Weld, 1994] Penberthy, J., and Weld, D. 1994. Temporal planning with continuous change. In *Proceedings of the Twelfth National Conference on Artificial Intelligence.* Menlo Park, Calif.: AAAI Press.

[Vere, 1983] Vere, S. 1983. Planning in time: Windows and durations for activities and goals. *IEEE Trans, on Pattern Analysis and Machine Intelligence* 5:246-267.

[Weld, Anderson, *k* Smith, 1998] Weld, D. S., Anderson, C. R., and Smith, D. E. 1998. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence,* 897-904. Menlo Park, Calif.: AAAI Press.