CRIKEY - A Temporal Planner Looking at the Integration of Scheduling and Planning

CRIKEY - A Temporal Planner Looking at the Integration of Scheduling and Planning

Keith Halsey and **Derek Long** and **Maria Fox**
University of Strathclyde
Glasgow, UK
keith.halsey@cis.strath.ac.uk

Abstract

For many temporal planning domains, the planning and scheduling problems are not tightly coupled and so can be solved separately. However, in some cases, where the problems do interact, this approach will fail. A domain is presented where this is the case. CRIKEY, a planner that separates out the logical and temporal reasoning, is introduced. It detects where they interact and the paper explains both how it detects them and also how, in these cases, CRIKEY solves the problems simultaneously. It will also look at CRIKEY as an architecture that uses a series of relaxations to find a plan. Preliminary results show its potential.

Separation of Problems

In temporal planning, classical planning moves closer to scheduling. This is demonstrated by PDDL2.1 (Fox & Long 2001), a language to express temporal planning problems. Actions, that can have a duration, must both be chosen for their logical and metric effects (planning) and then arranged in time so as not to break any constraints (scheduling). Conditions for the action are specified to hold at the start, at the end or over the duration of the action (these are called invariants). Effects are also specified to happen either at the start or at the end of the action.

This section briefly describes a system for separating out the logical and temporal reasoning with PDDL2.1 temporal domains. It was the precursor to CRIKEY and is described in more detail in (Halsey 2003) but is summarised in Figure 1. It takes a temporal PDDL2.1 planning domain and problem, and translates it into a classical PDDL (Ghallab *et al.* 1998) with a separate file to hold the temporal information. This is done using the translator from LPGP (Long & Fox 2003) that splits a durative action into three separate actions - a start action, holding the "at start" conditions and effects, an invariant checking action, that holds the invariants as conditions, and an end action, holding the "at end" conditions and effects. There are further dummy predicates to ensure that an end action is not placed in the plan without its corresponding start and invariant action. An example of this split is shown in Figure 2. The STRIPS problem is then solved by a classical planner to produced a totally ordered plan, from which a partially ordered plan is lifted, which in turn is scheduled using the information in the temporal file and a Simple Temporal Network to produce a temporal plan. FF (Hoffmann & Nebel 2001) was the classical planner used, although this could have been replaced by any classical planner.

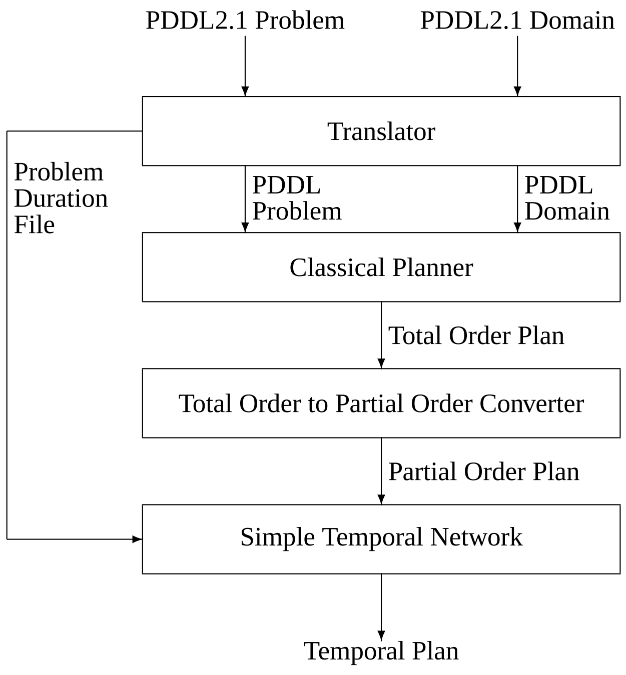

Figure 1: An Architecture for Separating Planning and Scheduling

This system separates out the planning and scheduling problems found in temporal problems (described in more detail in (Halsey, Long, & Fox 2003)) and solves each half separately. This has the advantage of making each problem on its own easier to solve and has proved successful in the domains of the IPC'02 (International Planning Competition

```
(:durative-action LOAD-TRUCK
   :parameters (?o -obj ?t -truck ?l -loc)
   :duration (= ?duration 2)
   :condition (and (over all (at ?t ?l))
                   (at start (at ?o ?l)))
   :effect (and (at start (not (at ?o ?l)))
                (at end (in ?o ?t))))
```

$$\Downarrow$$

```
(:action LOAD-TRUCK-START
   :parameter (?o -obj ?t -truck ?l -loc)
   :precondition (at ?o ?l)
   :effect (and (not (at ?o ?l))
      (load-inv ?o ?t ?l)))

(:action LOAD-TRUCK-INV
   :parameter (?o -obj ?t -truck ?l -loc)
   :precondition (and (at ?t ?l)
                (load-inv ?o ?t ?l))
   :effect (and (load-inv ?o ?t ?l)
                (iload-inv ?o ?t ?l)))

(:action LOAD-TRUCK-END
   :parameter (?o -obj ?t -truck ?l -loc)
   :precondition (and (load-inv ?o ?t ?l)
                    (iload-inv ?o ?t ?l)))
   :effect (and (in ?o ?t)
                (not (load-inv ?o ?t ?l))
                (not (iload-inv ?o ?t ?l)))
```

Figure 2: Translation of a Durative Action into Three Instantaneous Actions

2002) (Fox & Long 2002). The problem with this approach is where planning and scheduling interact, but this will be focused on later in the paper.

CRIKEY

This system was unified into one planner, called CRIKEY which is implemented in Java. However, there are some key differences in this re-implementation that shall now be explained. The planning technology at its heart is a forward chaining heuristic planner based very closely on MetricFF (Hoffmann 2002) making use of a relaxed plan length (for its heuristic) and helpful actions (to aid its action selection). MetricFF performs Enforced Hill-Climbing and on finding a plateau in the search space will resort to Breadth First Search to exit the plateau, not expanding any states that have a lower value than the current state. If this fails to find a higher state it will resort to complete search from the initial state. CRIKEY works in a similar manner, but will, on finding a plateau, perform Best First Search. This mean that it will explore the same states as FF to start of with, but on failing to find an exit, will try more states, expanding those which move away from the goal.

CRIKEY does away with the invariant action, separating durative actions into only a start and end instantaneous action (in fact, sometimes it does not even do this, but that will be discussed later). During the construction of the relaxed plan, if an invariant is achieved by the start effects, then it becomes a condition of the end action, else it becomes a condition of both the start and end action. Much like SAPA (Do & Kambhampati 2001), for each state the planner keeps a list of invariants that must hold and actions are not applicable if they delete one of these invariants. When a start action is chosen, its corresponding invariants are added to this list, and then when the corresponding end action is chosen, the invariants are removed from the list. This reduces the size of both the plan graph and the search space by a third. It also prevents an invariant being broken and re-achieved before an end action is chosen.

Similarly, a list is kept of actions whose start actions have been selected but not their corresponding end action. An end action is only considered if it is in this list. A final state is not reached until not only are the goals achieved, but also this list of actions is empty. This rids the need for dummy predicates to be put into the instantaneous actions, whilst still ensuring that an end action is not chosen without its corresponding start action and vice versa.

This list of semi-complete actions (a durative action where the start action has been chosen but not its corresponding end action) is also considered when the relaxed plan is extracted for the heuristic value. The goal is changed to contain dummy propositions which can only be achieved by putting in the end actions. This makes for an interesting ripple effect on plateauxs in the search space. If an action was required for its start effect, but not its end effect, then only the start action will appear in the relaxed plan. However, if this action is chosen then the end action will now appear in the relaxed plan, increasing its size by one, and turning otherwise flat plateau into having a ripple on it. This can be combated by setting the cost of the end actions to zero.

In fact, the cost of end actions is set to ϵ (the tolerance value) since otherwise there would be no incentive to put in an end action with no useful effects, and these must be in to make a valid plan. The start actions are given a cost which is equal to the duration of its durative action. During relaxed plan extraction, short actions are preferred to longer actions. This results in CRIKEY having a weak attempt at reducing the overall length of the plan.

Interaction of Planning and Scheduling

Whilst CRIKEY can perform competitively against other state of the art technology in the domains of IPC'02 (see (Halsey 2003) for results), the approach of separating the planning and scheduling has a major flaw, which does not occur in the IPC'02 domains. In these domains, all the solution plans could be sequentialised (the actions performed in a strictly ordered manner). That is to say that actions *could* happen in parallel if they didn't interfere, but that was a choice of the planner, and not enforced by the domain. However, it is possible to have domains where some actions *must* happen in parallel. An example of this is in the Match Domain (Figure 3). In this domain, the goal is to mend fuses. To mend a fuse (which takes 5 time units) you must have a hand free (i.e. you can only mend one fuse at once) and there must be light (provided by striking a match that lasts 8 time units). It should be obvious that in a problem instance where there are two fuses to fix, two matches are needed. A total of ten time units' worth of light is needed to mend the fuses sequentially, however a match only provides 8 time units of light. CRIKEY, as described above, will not realise this. Instead, a LIGHT_MATCH_START action will be chosen, then the actions necessary to mend both fuses, and then the LIGHT_MATCH_END action. Here an un-schedulable plan is produced and the temporal planner fails. In this case the sub-problems of planning and scheduling are more tightly coupled and are interdependent. The planning part critically effects the scheduling problem, and so in this case, the two cannot be separated.

```
(define (domain matchcellar)
(:requirements :typing :durative-actions)
(:types match fuse)
(:predicates (light)
   (handfree)
   (unused ?match - match)
   (mended ?fuse - fuse))

   (:durative-action LIGHT_MATCH
      :parameters (?match - match)
      :duration (= ?duration 8)
      :condition (and
         (at start (unused ?match))
         (over all (light)))
      :effect (and
         (at start (not (unused ?match)))
         (at start (light))
         (at end (not (light)))))

   (:durative-action MEND_FUSE
      :parameters (?fuse - fuse)
      :duration (= ?duration 5)
      :condition (and
         (at start (handfree))
         (over all (light)))
      :effect (and
         (at start (not (handfree)))
         (at end (mended ?fuse))
         (at end (handfree)))))
```

Figure 3: The Match Domain

A more in-depth look at where planning and scheduling interact in temporal planning (with particular reference to PDDL2.1) can be found in (Halsey, Long, & Fox 2003). A very brief summary follows. Interactions between scheduling and planning occur where there are durative actions (called *content* actions) that must be executed in the time that another sequence of actions (called *envelope* actions) execute. In these cases, the minimum length of time for the content actions must be less (or fit into) the maximum total length of time for the envelope actions. If this is not the case, then an un-schedulable plan is produced.

During this discussion, it would be perhaps fairer to refer to "envelopes" as "potential envelopes". No effort (at this stage) is made to examine the domain further to see if the content actions could in fact be placed outside their envelopes, or indeed, if there are any content actions in the domain for the envelope. However, on this last point, whilst this could lead to extra envelope reasoning being performed, it does not alter the completeness of the algorithm. The envelope will in effect be empty. The work presented is conservative in this aspect, with the aim being to only produce schedulable plans. For conciseness, during the discussion, "potential envelopes" are simply referred to as "envelopes".

All this reasoning is particular to PDDL2.1, and its corresponding model of time. Logical change in PDDL2.1 happens only at the start and end of actions. However, it is still possible, through simple compilation of the domain and the use of dummy actions and predicates, to model more complex models of time (including change happening part way through an action) and also more complex temporal constraints (for example, enforcing actions to overlap to a degree). The reasoning that follows does not explicitly tackle these issues, but if the constraint is compiled into PDDL2.1, then the reasoning will hold (since now all change once again happens at the start and end of actions) and a valid, schedulable plan can be produced.

Detecting and Acting with Envelopes

The basic principle behind CRIKEY is to separate planning and scheduling where possible, and solve the two problems together only where this is strictly necessary. This section describes how CRIKEY distinguishes between the two situations, and how it acts in each case. Both eventualities affect two places in the algorithm. The first is during the calculation of the heuristic (i.e. during the construction of the relaxed plan graph and the extraction of the relaxed plan) and the second is during action selection. There are two affects of envelopes. One is to allow the further compression of durative actions, and the second, that only affects action selection, is to integrate scheduling into the algorithm at certain points. It should be noted throughout the following reasoning that CRIKEY does not allow negative conditions at any point in a durative action.

Compressing Actions in the Relaxed Plan Graph

During the calculation of the heuristic, delete effects are ignored, and so can also be ignored in the reasoning here. In this part of the algorithm, a durative action is considered an envelope action if there are any at end conditions not met by an at start add effect. If this is not the case (i.e. $(cond_{end} \setminus add_{start}) = \emptyset$) then rather than splitting the durative action up into two separate start and end actions, it is treated as a single condensed action. The at start and at end add effects are combined to form the effects of the single action. The conditions to the new action are the at start conditions of the durative actions combined with any invariants of the durative action which are not met by the at start add effects. More precisely (where D is the Durative Action and A is the newly condensed action):

$$((D_{cond_{end}} \setminus D_{add_{start}}) = \emptyset) \Rightarrow$$
$$A_{cond} = D_{cond_{start}} \cup (D_{cond_{invs}} \setminus D_{add_{start}})$$
$$A_{add} = D_{add_{start}} \cup D_{add_{end}}$$

This action A is then treated as any other action in the relaxed plan graph. Suppose there are two planning graphs, one with the actions split and one where it is not. The fact layer $n+1$ (n being where the start action can first be applied) in the graph where the action has been split is the same as layer n in the graph where the condensed action has been used and thus the compaction is valid. Since both the end action in action layer $n+1$ and the start action in layer n must be chosen, the algorithm saves time by effectively choosing both actions (and so both set of add effects) at once. The graph is also now smaller (saving both time in construction and extraction) as it is (potentially) one layer shorter. In the case where all actions can be condensed, the graph is half the size as when the actions are not condensed (or a third of the size of the original system also had invariant actions).

The actions cannot be condensed if they have at end conditions since to meet those conditions might require the use of at start add effects (or rather, an action that achieves the at end conditions, might itself have conditions that are met by the at start add effects) as in Figure 4. A cannot be put in the RPG without B and B cannot be put in without A.

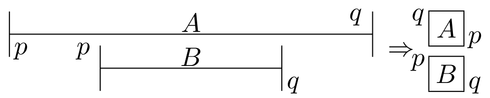

Figure 4: Example of failure of compressing the action for a RPG

Compressing Actions During Action Selection

Similar reasoning to the above can be used in detecting envelopes during action selection, but now delete effects must also be taken into account. During action selection, if you put in a start action of a durative action, then at some point you must put in its end action. Therefore, when a start action is put in, the end action may as well be put immediately afterwards. However, this logic fails where an action could be applied *only* in the state after the start action but not in the state after the end action and also not in the state preceding placing the state action in (as would happen in Figure 5).

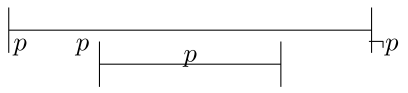

Figure 5: Example of failure of compressing the action for state selection

We shall name three states, $s1$, the state immediately before the application of the start action, $s2$, the state immediately after the application of the start action, and $s3$ the state immediately after the application of the end action. An action applicable in $s2$ and not in $s1$ must have been achieved by the at start add effects (since there are no negative conditions, it could not have been achieved by an at start delete effect). Taking it further, there are no actions that could be applied in $s2$ and not in $s3$ which could not have been applied in $s1$, apart from those achieved by the at start add effects and then deleted by the at end delete effects. The other case where actions cannot be applied in immediate succession is the same as for actions in the relaxed plan graph, that is that there isn't an at end condition which isn't met by an at start add effect. If both of these cases do not hold, then both the start and the end action can be applied one after the other without any risk to completeness, with no further search needed. More precisely, this can happen only where:

$$\begin{gathered}((cond_{end} \setminus add_{start}) = \emptyset)\\ \wedge((add_{start} \cup del_{end}) = \emptyset)\\ \wedge((del_{start} \cap cond_{end}) = \emptyset)\end{gathered}$$

The applicability of both actions in state S must be determined at the same time. They are applicable if:

$$\begin{aligned}&cond_{start} \cup (cond_{end} \setminus add_{start}) \cup cond_{invs} \subseteq S_{facts}\\ &\qquad\qquad\wedge(del_{start} \cup del_{end}) \cap S_{invs} = \emptyset\end{aligned}$$

It is possible that another action, different to the envelope action, could have achieved the content action. This is trivial to test for in forward search since that proposition will already be true, and in this case the content action will not have to go inside the envelope.

Ensuring Schedulabilty

Exactly where the split actions cannot be applied one after the other is where the planning and scheduling problems interact and the two problems cannot be solved independently. If planning is done with no consideration for scheduling, then the risk is run of producing an un-schedulable plan. To solve this, whilst still leaving the scheduling until after planning, potenetial envelopes must be detected during planning and then scheduling only done for them. An envelope is an action where the split actions cannot be applied one after the other:

$$\begin{gathered}((cond_{end} \setminus add_{start}) \neq \emptyset)\\ \vee((add_{start} \cup del_{end}) \neq \emptyset)\\ \vee((del_{start} \cap cond_{end}) \neq \emptyset)\end{gathered}$$

To recap, there must be time for a sequence of content actions to be executed whilst an envelope action executes. Any time a start action from an envelope is selected, a mini-scheduler is associated with that action. Any potential content action then selected after that is tested to see if it must fit in between the envelope's start and end action, and also ordered with regard to any other content actions in the envelope. If so, then it is tested to see if there is enough time to execute the action. If so, only then is it selected and added to the contents of the envelope, or else that action is not applicable in the current state. Once the envelope's end action is selected, then the mini-scheduler for the action is discarded. Each envelope has associated with it, its start action, its end action, a Simple Temporal Network, and a list of content actions that must be inside the envelope.

This Simple Temporal Network need only test for consistency and so uses Bellman-Ford's algorithm to test for negative cycles from appropriate sources (those nodes where there have been edges added to it). An algorithm for deciding whether an action is applicable in a state including these envelope actions is described in Figure 6. By pruning out actions that are not applicable, all plans produced are schedulable.

```
1. Check $A_{cond}$ are satisfied.
2. Check $A_{del}$ do not delete any currently open invariants.
3. If A is a start of an envelope
   (a) Create a new envelope for $A$ and add to list of envelopes.
4. Else If A is an end of an envelope
   (a) Remove $A$'s envelope from the list of envelopes.
5. For Each envelope $E$ in the list
   (a) Get orderings for $A$ in $E$.
   (b) If no orderings, return true.
   (c) Add orderings to the STN.
   (d) return the consistency of the STN.
```

Figure 6: Algorithm to decide whether an action A is applicable

To convert to the partial order plan, only those actions that interact need be ordered. These arise in two cases; firstly, where one action achieves another, and secondly, where one action threatens another by deleting a precondition. The partial order lifter used is a modified version of the one described in (Moreno *et al.* 2002). It is a greedy algorithm that works backwards through the totally ordered plan. All threatening actions that appear before an action in the totally ordered plan, are constrained to appear before it in the partially ordered plan, and the closest action that achieves it in the totally plan, must be ordered before it in the partially ordered plan. Although this will not find an optimal plan, that is to say one that exploits all concurrency possible, it is complete and sound. This is modified to also be able to cope with metrics, such that all consumers of a resource are ordered before an action that has a greater than precondition using that resource, and enough producers are also ordered before that action. (The producers and consumers reverse roles in the case of a less than precondition).

A Simple Temporal Network (**?**) with Floyd-Warshall's algorithm run once computes the actual timings of the actions to produce a valid temporal plan.

Example In the case of the match domain (Figure 3), assuming it is part of a bigger domain, CRIKEY will search forward ignoring temporal information. When it comes to put in the start action to the light match action (an envelope), it will instantiate a new mini-scheduler. It will then test to see if a fix fuse action need go in this mini-scheduler, and if so, if it is consistent. Indeed, it fits, so the action is applicable and choosen for the plan. It will then test the second fix fuse action. This is not consistent with mini-scheduler (there is not enough time left to fix it before the match runs out), so cannot be put in the plan. (If the fuses could be fixed in parallel, then this second action would be consistent and so chosen). The end of the light action could then be chosen and the mini-scheduler and envelope would be closed. CRIKEY would then go on to either light a second match (and so start a new mini-scheduler) or solve another part of the problem. In this way a schedulable plan is produced.

Bigger Envelopes

Rather than having a content action achieve the end condition to the envelope action, a further envelope action could achieve it in its start effects. In this case, the envelope is from the start of the first envelope to the end of the second envelope and the mini-scheduler is not closed until this second envelope action finishes (i.e. the end action is chosen). There can, of course, be many envelopes making up the big envelope. It is also important to note that content actions can themselves be envelope actions.

Relaxations

An alternative way of looking at CRIKEY is as a series of relaxations is detailed in Figure 7. The first relaxation is in the relaxed plan where delete effects are taken out of the problem. This guides the search both in the heuristic distance to the goal state and also by providing "helpful actions". Where this relaxation fails, it resorts to full search without helpful actions. The second relaxation is in the non-temporal sequential plan, which forms a skeleton for the temporal parallel plan, where temporal effects are taken out. Relaxation also occurs where the durative actions are condensed into instantaneous action where there is no advantage in splitting them. However, where the relaxation is too severe (i.e. when leaving out the temporal information leads to no solution) the relaxation is tightened up, and the temporal information taken into account. The skill of CRIKEY is in recognising where this is the case.

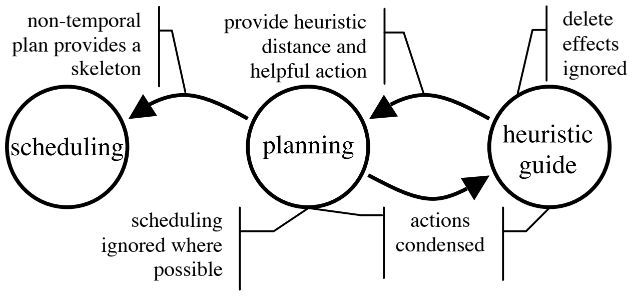
non-temporal plan provides a skeleton
provide heuristic distance and helpful action
delete effects ignored
scheduling
planning
heuristic guide
scheduling ignored where possible
actions condensed

Figure 7: Gradual Relaxations of CRIKEY

Preliminary Results

A more comprehensive set of results, comparing the old system based on FF described at the beginning of this paper with other planners on some problems from IPC'02 can be found in (Halsey 2003). Figures 1 & 2 compare this old system with the newly implemented CRIKEY and its differences and also with SAPA (Do & Kambhampati 2001). It would be expected that CRIKEY would perform polynomially better over the FF system, since it compresses all the actions in this domain, and so has a third of the actions. However the results show the opposite, a polynomial decrease in performance. Primarily, this is put down to a different implementation language (Java rather than C++). However, CRIKEY is more expressive, not only being able to tackle numerical problems but of course also domains where the planning and scheduling interact more. This carries an overhead which may also account for the decrease in performance. CRIKEY does perform quicker on problem 10. This is because whereas the FF System resorts to planning from the initial state if it finds a dead end, CRIKEY will start complete search from where it entered the plateau which led to the dead end.

Table 1: DriverLog SimpleTime Times

pfile	FFSystem	CRIKEY	SAPA
2	620	1150	7500
4	600	1880	8550
6	740	5990	2080
8	2580	14350	2180
10	2510	840	1560

Table 2: DriverLog SimpleTime Plan Quality

pfile	FFSystem	CRIKEY	SAPA
2	100.03	126.07	104.09
4	89.03	121.04	98.12
6	109.04	114.06	64.07
8	51.03	111.04	75.08
10	91.04	71.02	36.05

CRIKEY is also compared against SAPA since this planner is also implemented in Java and can plan with metric values and find plans where the logical and temporal constraints interact. As can be seen, generally SAPA is slower but finds better quality plans. This is to be expected since it performs A* search, therefore searching more states, but will most likely find a better plan. It is quicker on problem 8 since this is where CRIKEY's forward heuristic search gets stuck in a dead end and it must resort to complete search.

A domain has been written called the lift-match domain to test CRIKEY's abilities. In it, electricians must travel round a building fixing fuses as in the match domain. It not only contains the problem where the scheduling and planning interact, but also a simple lift scheduling problem, a logisitics problem as well as resource management all embedded in. If an electrician has already lit a match in a room, then there is enough light for another electrician to fix a fuse by. Time Initial Literals (as in PDDL2.2 (Edelkamp & Hoffmann 2003)) can be introduced by having fuses blowing at certain times. As of yet this domain has not been tested extensively with other planners, but CRIKEY can solve instances whereas it is thought that some other planners may not be able to.

CRIKEY hopes to compete fully in the International Planning Competition 2004, and so the results from that will hopefully give a better idea of CRIKEY's performance compared with other planners.

Conclusions and Further Work

CRIKEY shows that it is possible to separate out temporal and logical reasoning, whilst combining them where necessary. CRIKEY performs the reasoning necessary to do this. It also demonstrates that it is possible to perform a series of relaxations in order to find a plan, and tighten these relaxations where necessary. It is hoped that these themes will be explored further in the near future. The scheduler is of particular interest since at the moment it only performs a greedy search. It is thought that the metric reasoning and logical reasoning can also be separated out in the scheduler, with the scheduler also performing some search to improve the quality of the plan, depending on the metric of the original problem.

References

Do, M. B., and Kambhampati, S. 2001. Sapa: a domain-independent heuristic metric temporal planner. In *Proceedings from the 6th European Conference of Planning (ECP)*.

Edelkamp, S., and Hoffmann, J. 2003. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical report, Fachbereich Informatik, Germany and Institut für Informatik, Germany.

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, UK.

Fox, M., and Long, D. 2002. The third international planning competition: Temoral and metric planning. In *Proceedings from the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02)*, 115–117.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—the planning domain definition language.

Halsey, K.; Long, D.; and Fox, M. 2003. Isolating where planning and scheduling interact. In *Proceedings from the 22nd UK Planning and Scheduling Special Interest Group (PlanSIG'03)*, 104–114.

Halsey, K. 2003. Temporal planning with a non-temporal planner. In *Doctoral Consortium at the 13th International Conference on Automated Planning and Scheduling (ICAPS)*, 48–52.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J. 2002. Extending FF to numerical state variables. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*, 571–575.

Long, D., and Fox, M. 2003. Exploiting a graphplan framework in temporal planning. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*, 52–61.

Moreno, D.; Oddi, A.; Borrajo, D.; Cesta, A.; and Meziat, D. 2002. Integrating hybrid reasoners for planning and scheduling. In *Proceedings from the 21st UK Planning and Scheduling Special Interest Group (PlanSIG'02)*, 179–189.