

The Workings of CRIKEY - a Temporal Metric Planner

Keith Halsey

University of Strathclyde
Glasgow, UK
keith.halsey@cis.strath.ac.uk

Abstract

Described here is the temporal metric planner CRIKEY as it competed in the International Planning Competition 2004. CRIKEY separates out the planning and scheduling parts of temporal planning problems, and detects where these two sub-problems are too tightly coupled to be separated completely. In these cases it solves the sub-problems together. The domains of the competition are looked at to see where these interactions occur.

Introduction

CRIKEY is a forward heuristic search planner based closely on MetricFF (Hoffmann 2002) and implemented in Java1.4. In a similar fashion to MIPS (Edlekamp & Helmert 2000), it separates the planning and scheduling where it can, however it solves the two problems together where such a relaxation will fail. It is this combining of the problems only where necessary and the reasoning associated with it that distinguishes it from other similar planners (and where the focus of the research lies). It can detect these cases in the domain and act accordingly. I am only interested in where the interaction and separation of sub-problems will prevent a solution being found, and not where this separation leads to an inferior quality of solution. CRIKEY is complete and sound but not optimal (either in time or the specified metric). It will however make an attempt to minimise the number of actions in a plan.

Capabilities

CRIKEY was written to work with the PDDL2.1 (Fox & Long 2001) models of metrics and time. It can deal with both temporal aspects (i.e. durative actions) and metrics resources. More formally, it can parse and plan with PDDL domains with the `:typing`, `:fluents`, and `:durative-actions` requirements. Unfortunately, currently it can not make use of any of the ADL constructs or the new language features (namely, timed initial literals or derived predicates).

Copyright © 2004, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Architecture

The architecture of CRIKEY is shown in Figure 1. It first looks at the domain for where planning and scheduling could potentially interact. Then it performs forward heuristic search using a relaxed plan graph. The mini-scheduler makes sure that a schedulable plan is passed into the scheduling phase. This consists of lifting a partial order plan from the totally ordered plan, and then turning this into a temporal plan. Crucially, there is no feedback from the scheduling phase to the planning phase, therefore the planner must produce a plan that the scheduler can schedule.

Technical Details

Planning

CRIKEY finds a plan through forward heuristic search similar to FF (Hoffmann & Nebel 2001). During planning, temporal information is ignored. The search strategy is enforced hill climbing, that is, once a better state is found, search proceeds from that state without backtracking. Best first search is used on plateaus, where all neighbouring states are no improvement on the current state. If enforced hill climbing fails, best first search is attempted from the initial state. This is complete and so theoretically should find a plan.

The heuristic value is the length (number of actions) of a relaxed plan where the delete effects are ignored. The relaxed plan is from the current state to the goal state and is easily extracted from a relaxed planning graph.

As in FF, only helpful actions are considered in the enforced hill climbing. Helpful actions are actions which appear in the first layer of the relaxed planning graph and are also in the relaxed plan.

Scheduling

A greedy algorithm (Moreno *et al.* 2002) works backwards through the totally ordered plan finding causal links between the starts and ends of actions to form a partially ordered plan. Links are either \leq or $<$ (in which case a minimum value equal to the tolerance value must separate the two end points). These are put into an STN upon which Floyds-Warshalls Algorithm is to calculate the actual time of the actions in the partially ordered plan.

The algorithm must not only look for orderings based on logical conditions, but also for orderings due to metric con-

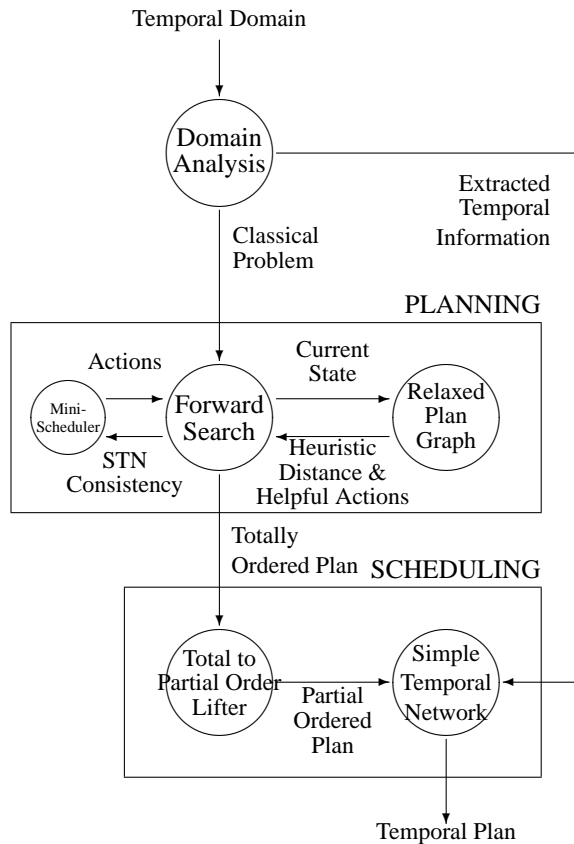


Figure 1: Architecture Overview of CRIKEY

straints. For a $>$ or \geq resource constraint, just enough producers of that resource are ordered before it, assuming that all consumers that precede it in the totally ordered plan, occur before it in the partially ordered plan. The same is true for $<$ or \leq conditions, apart from the roles of consumer and producers are reversed. Whilst this is conservative, it must be sound as the totally order plan is correct (at worst, the partial order will be the same as the total order).

The next section details how it is impossible to produce an unschedulable plan.

Interactions

In cases where the planning and scheduling interact, precautions must be made to ensure that a plan is not produced which is unschedulable. This can happen where the actions *must* happen in parallel (as opposed to the more common case where actions *can* happen in parallel if they do not interfere). That is to say, one or more actions (called “content actions”) must happen whilst another (the “envelope action”) is executing. If there is not enough time to execute the contents during the envelope, then an unschedulable plan is produced.

These cases are detected in advance by looking for “potential envelopes” - actions which allow other actions only

to happen during their duration. These happen where:

$$\begin{aligned}
 &(end_{cond} \setminus start_{add} \neq \emptyset \wedge start_{add} \setminus end_{cond} \neq \emptyset) \\
 &\quad \vee start_{del} \cap end_{cond} \neq \emptyset \\
 &\quad \vee add_{start} \cap del_{end} \neq \emptyset
 \end{aligned}$$

We shall name three states, s_1 , the state immediately before the start of the action, s_2 , the state immediately after the start, and s_3 the state immediately after the end of the action. An action applicable in s_2 and not in s_1 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 s_2 and not in s_3 which could not have been applied in s_1 , apart from those achieved by the at start add effects and then deleted by the at end delete effects. Alternatively, an action could be achieved by the start effect, and the effects of this action needed to achieve the end conditions. They are called potential envelopes since (at the moment) there is no effort to find out if there are any content actions that must go in these envelopes.

As stated, where there are potential envelopes, there is the potential to produce an unschedulable plan. To avoid this, envelope action are split into two separate actions, a start action containing the start conditions and effects, and an end action containing the end conditions and effects. Invariants become conditions of the end action, and, if not achieved by the start effects, also of the start action. An end action cannot be applied until its corresponding start action is in the plan, and a plan is not valid until all the start actions in the plan also have their corresponding end actions in the plan.

On putting a start action into the plan, a mini-scheduler is associated with this action. This mini-scheduler consists of a Simple Temporal Network, a set of content actions (initially empty) and a set of orderings between these actions. The mini-schedulers use the same algorithms as the main scheduling part of CRIKEY. Any (content) actions which are now considered, must be checked against this mini-scheduler to ensure that if they must go in the envelope, the STN is consistent (that is to say that there is enough time to execute the action). If not, then the action is not considered applicable, and that branch is removed from the search space. When the envelope’s end action is chosen, the mini-scheduler is then discarded. Figure 2 is pseudo-code for the mini-scheduler. As can be seen, invariants are protected whilst an envelope’s start has been chosen but not its end action. No other action may delete these invariants until that action has completed.

Competition Domains

Unfortunately, none of the domains in the 2004 competition in their purest form (that is, without the new features compiled out) contained any envelopes (i.e. no actions *had* to happen in parallel) and so in all problems the planning and scheduling were relatively loosely coupled. This means that CRIKEY could not show off its mini-scheduling capabilities to cope with these situations. It is hoped that after the competition, the other competing planners will become available and it will be possible to compare them with CRIKEY on domains which do contain such situations.

1. Check A_{cond} are satisfied. If not, return false.
2. Check A_{del} do not delete invariants in the list of invariants. If not, return false.
3. If A is a start of an envelope
 - (a) Create a new mini-scheduler for A and add to list of mini-schedulers.
 - (b) Add A 's invariants to the list of invariants.
4. Else If A is an end of an envelope
 - (a) Remove A 's mini-scheduler from the list of mini-scheduler.
 - (b) Remove A 's invariants from the list of invariants.
5. For Each envelope E currently open
 - (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 2: Algorithm to decide whether an action A is applicable

Envelopes were present in versions of the domains where timewindows and deadlines had been compiled down from PDDL2.2 to PDDL2.1. These envelopes are present in the newly created dummy actions to enforce the constraints and lasted the length of the plan. As the envelope lasts the length of the plan, the mini-scheduler for each dummy action is active throughout the planning process. This is highly inefficient and not what the mini-schedulers are designed to solve. However, it still makes sure that an unschedulable plan is not passed to the scheduler.

Since there were no domains particular to CRIKEY's designed purpose and strengths, not much development of CRIKEY was performed whilst the competition was running, except to correct bugs in the code and parser. It is thought that not being able to handle ADL was not such a disadvantage as CRIKEY would probably have only performed an equivalent compilation internally.

References

- Edlekamp, S., and Helmert, M. 2000. On the implementation of mips. In *Proceedings from the 4th Artificial Intelligence Planning and Scheduling (AIPS), Workshop on Decision-Theoretic Planning*, 18–25. Breckenridge, Colorado:AAAI-Press.
- Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, UK.
- 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.
- Moreno, D.; Oddi, A.; Borrajo, D.; Cesta, A.; and Meziat, D. 2002. Integrating hybrid reasoners for planning and