Isolating Where Planning and Scheduling Interact

Keith Halsey, Derek Long and Maria Fox
University of Strathclyde
Correspondence to: keith.halsey@cis.strath.ac.uk

September 2003

Abstract

In many cases it is possible to separate out the causal and temporal reasoning needed for temporal planning into classical planning and scheduling. However, where the two problems are tightly coupled, that is, where there are temporal constraints encoded in the problem which will affect the choice of action, this strategy can lead to producing un-schedulable plans. This paper analyses where such cases can arise through encoding temporal constraints as durative actions and then examining where the constraints can be broken. It is found that it is not always possible to translate the problem to loosen the coupling. One option still open is to leave the problems connected only where necessary. The paper looks at how this could be done and the advantage of taking this approach.

1 Introduction

Temporal Planning is the same problem as Classical Planning, but with time introduced to the problem. There are many different models of time in Temporal Planning, but the most common is for the actions to have duration. PDDL2.1 [4] is a domain and problem description language that uses such durative actions. Durative actions can have conditions that must be met at the start or end points of actions, effects that happen at either the start or end points of actions and invariants, propositions that must hold for the duration of the action. The duration is either fixed or a function of the action's parameters.

Temporal Planning can also be seen as the combination of Classical Planning and Scheduling. Whereas classical planning is the problem of selecting actions to meet a given goal without breaking any logical constraints, scheduling is the problem of deciding when to perform those actions so as not to break any temporal or resource constraints. Both of these are well known hard problems, and combining them can only make the problem harder, since now the logical, temporal and resource constraints must all be met simultaneously.

This paper looks at the possibility of separating the two sub-problems to solve them separately. This has the advantage that the two sub-problems should be easier to solve independently, where

constraints relevant to the other problem may safely be ignored. Further more, specialised solvers that have already been written can be used in solving the sub-problems. As when separating any problems, difficulties occur where the problems interact (that is, they are tightly coupled). It is only easier (and potentially possible) where the large problem (in this case, temporal planning) can be decomposed and the sub-solutions (the plan and schedule) can be successfully re-composed.

HybridSTAN [3] is a planner that finds cases of route planning problems in planning problems and then uses a specialised solver to solve these cases separately. It is successful even where the embedded route planning problem is disguised. Further work on STAN [6] looks in greater detail at how sub-problems can be abstracted from a planning domain and solved independently. It analyses cases where this is possible and the relationships between sub-problems. There are three main groups of relationship recognised. Firstly, there is the case were there are single and multiple independent sub-problems, and here the integration of the planner and sub-solver is simple. The solutions to the sub-problems do not rely on each other. Secondly, there could be a hierarchical sub-problem relationship. In this case, there is a hierarchy of sub-problems, where one sub-problem may in turn have embedded in it another sub-problem to be solved. The solution of one problem will rely on the solution of another, but not in reverse. Thirdly, sub-problems in a domain may be interdependent, and the problems cannot be resolved into a strict precedence ordering, as in the case of a hierarchy. The solution of one problem will rely on the solution to another *and* vice versa. This case, where the sub-problems are tightly coupled, represents a much harder integration of the planner and sub-solvers.

RealPlan [11] is a planner that separates the causal reasoning from the resource reasoning. There are two versions, the first, RealPlan-MS (Master-Slave) and the second, RealPlan-PP (Peer-to-Peer). The difference is in how the scheduler of the resources and the planner interact. In both cases, the resources are abstracted out of the domain and translated into a CSP (Constraint Satisfaction Problem), which is then solved by a specialised CSP solver. In the Master-Slave scenario, should the scheduler fail to find a solution to the current context, then that partial plan is not pursued any further. In this version, where all the allocation policies lead to failure, it implies that the causal reasoning and the resource reasoning were, in fact, tightly coupled. In this case, the planner resorts to traditional planning methods where the resource reasoning is not abstracted out. However in the peer-to-peer relationship, the causal reasoning is also translated into a CSP (from a plan graph). Both CSP problems can be solved simultaneously and so should the scheduler not find a solution it can tell the planner why not (i.e. what constraints are broken) and the planner can act accordingly.

Most temporal planners, for example SAPA [2], TPSys [7] and LPG [8], do not separate out the temporal and causal reasoning in temporal planning. The rest of this paper investigates how this can be done, the relationship between planning and scheduling in different domains and the difficulties that arise from trying.

2 Separation of the Problems

A temporal planning system designed to separate out the temporal and logical aspects has been implemented and is detailed in [9]. Figure 1 is an abstracted overview of how the system works. Firstly, a temporal planning domain and problem is passed through a translator which takes out

the temporal aspects, converting it to an equivalent STRIPS-like domain that preserves all the key temporal relationships. Durative actions are split into three instantaneous actions, one representing the start of the action, one, the end and one, the invariant. It stores the duration of the actions in a separate file. The translated problem is passed through a classical planner, in this case FF [10]. This is where the 'hard' work is done, but should be easier without the temporal information. The totally ordered sequential plan is passed through a program that produces a partially ordered plan, allowing actions that can be executed together to happen concurrently. The partial ordering, along with the duration file created by the translator, are passed into a program that uses a Simple Temporal Network [1]. This schedules the plan by calculating the relative and actual timings of the actions, to produce a valid temporal plan.

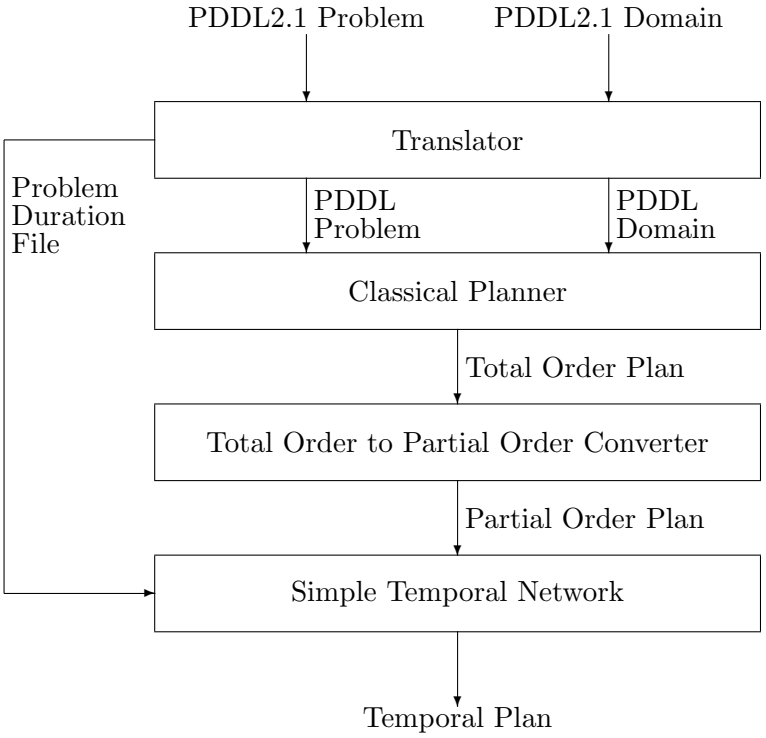

Figure 1: An Architecture for Separating Planning and Scheduling

The affect of this architecture is to separate the temporal planning problem into its two component halves, that of the planning problem and the scheduling problem. This architecture can solve the problems in the International Planning Competition 2002 (IPC'02) [5] domains, so long as FF can find a plan. Since FF is theoretically complete, even if practically it can't solve all the problems due to time and memory constraints, this would seem a good strategy to use. By solving two smaller sub-problems, rather than one large problem, the whole problem in theory should be easier (as discussed previously). However, there are two main disadvantages to this approach, both stemming from the same root: The classical planner is lacking temporal knowledge and so cannot make any decisions based on it. The first problem is that the planner can produce plans which the scheduler cannot schedule, and the second is that the system cannot intelligently optimise the length of the plan. These two problems are looked at in more detail in the next section.

3 Analysis of the Problems

Looking again at the domains used in IPC'02 [5], they all have a common structure that permits this architecture to work. All solutions to the problems (i.e. temporal plans) could be stretched out such that each action happens sequentially. Of course, in most cases this would lead to bad quality plans, and there is the problem of finding out which actions can occur concurrently without interfering, but this is not a hard problem to solve. What is important is that this concurrency is a choice made by the planner in the production of the plan, and not enforced in order to find a solution, as the logical outcomes of the actions are only affected by the sequence they are carried out in. Some actions *can* happen concurrently and the scheduling and planning sub-problems are loosely coupled in a hierarchical fashion (see Section 1).

There are other cases, not represented in the domains of IPC'02, where the plans cannot be sequentialised, but rather some actions *must* happen concurrently. An example of this is given in Figure 2. 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. The architecture described above will not realise this. Instead, the classical planner chooses a LIGHT_MATCH_START action, 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 are more tightly coupled and are interdependent.

The second instance where this architecture falls down through the planner not having any temporal knowledge is that it cannot attempt to minimise the total duration of the plan, only the number of actions performed. Suppose there is a third action, LIGHT_BIG_MATCH, in the domain that is the same as LIGHT_MATCH, except that it has a duration of 10 time units. In the case where there is one fuse to fix, it is obviously best to use one small match, as the plan will be shorter in duration (although the number of actions is the same). Where there are two fuses, it is now better to use one big match rather than two small ones since again the plan will be shorter. In the case of three fuses it is now best to use two small matches again, as one big match will not provide sufficient light but any other combination of matches will take longer to burn. The current architecture will not be able to make any of these decisions, and these are simple cases. It does not take much imagination to envisage scenarios where the temporal aspects of the actions interact in a much more complicated manner.

The first of these problems, that of producing un-schedulable plans, is more serious as it means plans cannot be found regardless of quality, and will be focused on for the rest of this paper.

4 Temporal Constraints

The un-schedulable plans come from temporal constraints in the problem that are not met. In PDDL2.1, temporal constraints are not represented explicitly, but rather implicitly, using other,

```
(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 2: The Match Domain

potentially dummy, durative actions as these are the only way to get temporal information into the plan. What follows is a review of possible constraints that can be represented in PDDL2.1 and the different ways that these constraints can be expressed. All other constraints that do not use disjunctions are simply specialised cases of these. Figure 3 i) and ii) show how the constraint $x - y \leq b$, which semantically represents the maximum time by which x follows y, can be enforced using a dummy action[1]. Figure 3 iii) and iv)shows how the constraint $x - y \geq b$ can be encoded, and represents the minimum time that x follows y by.[2]

These figures have been arranged in a fashion that should make obvious the similarities both within the different representations of the same constraint, and also the similarities in representing the different constraints. One interesting point to note here is the direction of the arrows. Regardless of the form used, when expressing a maximum time between actions, the ordering is from the start of the dummy action to A, and from B to the end of the dummy action, whereas when expressing a minimum time, regardless of the form used, the ordering is from A to the start of the dummy action and then from the end of the dummy action to B. This is all strangely reminiscent of a Simple Temporal Network [1], where in the graph a directed edge in one direction has one meaning (maximum time difference), but in the other has another (minimum time difference). The weights

[1]Here $x - y \leq b$ can be rearranged as $y - x \geq -b$, which of course means exactly the same. Other constraints can be equally rearranged, however, since an action cannot have a negative duration, all constraints are kept in the form that keeps b non-negative (regardless of whether it is an upper or lower limit).

[2]For the durations of the dummy actions in Figure 3, ϵ (or tolerance value) would have to be taken into consideration. However, for ease of understanding it is currently omitted.

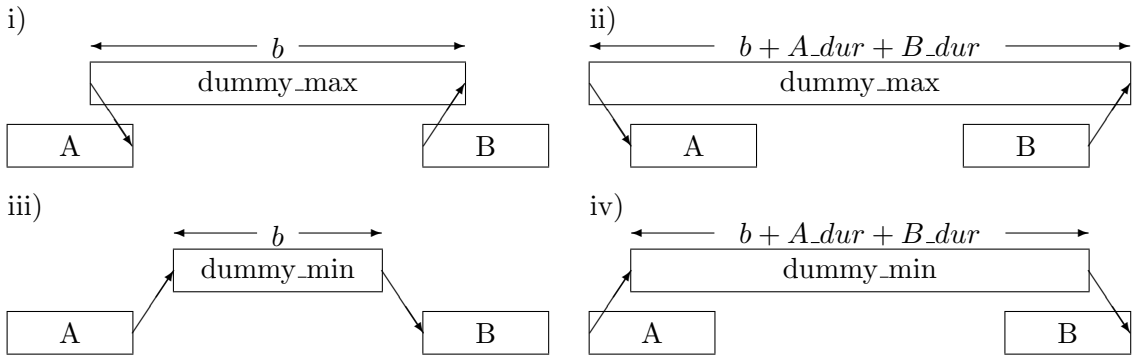

Figure 3: Expressing a Maximum Minimum Elapsed Time Between Actions in PDDL2.1

of the edges are provided by the duration of the dummy action. The different representation can be "mixed and matched" within themselves, as shown in Figure 4 using a compacted notation. The orderings (or precedence) of the actions is forced through logical constraints. For example, one end point action achieving a condition for another end point action, or one end point action deleting a condition for another end point action. There are other examples (which include reasoning with invariants) which are not detailed here, suffice to say that how these orderings are achieved does not affect the effect they have. This applies to the whole paper.

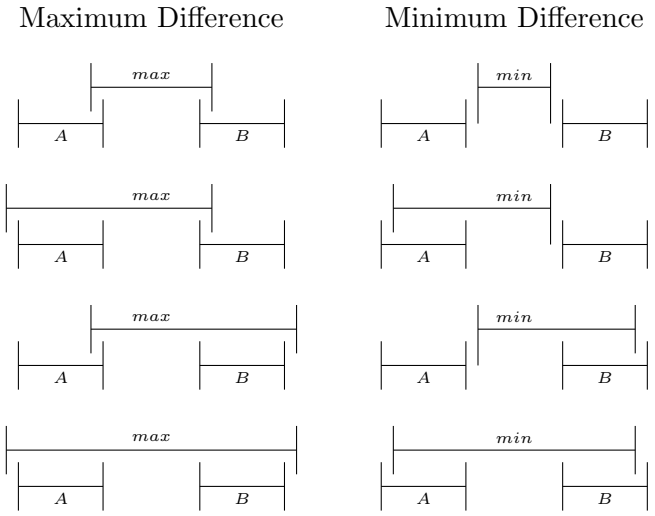

Figure 4: Possible Combinations of Representing the Same Constraint

With the maximum times, there is no minimum, so there is nothing to stop B in fact being ordered before A, and of course with the minimum times, B could happen infinitely after A without breaking the constraint. The more interesting cases occur when both a maximum and minimum time occur, i.e. where the constraints are combined to the form $b_1 \leq x - y \leq b_2$. To form these constraints, the minimum and maximum constraints are simply combined in any combination, as in Figure 5. Of course, for this to be possible, the duration dummy_min must be less than or equal to the duration of dummy_max.

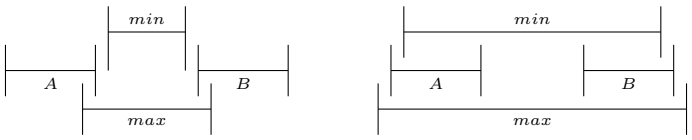

Figure 5: Expressing both Minimum and Maximum Time Between Actions in PDDL2.1

5 Translation of the Domain

Let us consider another domain: the breakfast domain which involves making a cup of tea. There must be a maximum time between boiling the water and pouring it into the mug (or else the water cools, and tea cannot be made with cold water). This could be expressed in two ways; either by the first *clipping* method, or the second *enveloping* method (as seen in Figure 6). The ordering between the boiling of the water and pouring it is equivalent to a minimum time of zero.

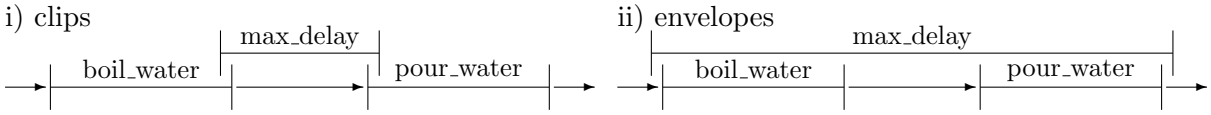

Figure 6: Two Possible Equivalent Representation of the Breakfast Domain

Now it should be obvious that these two representations are equivalent. However the architecture described in Section 2 would not have trouble with the former of these two where "clips" are used whereas, with the "envelope" method, it could produce an un-schedulable plan. This would occur where, as with the light match action (see Section 3), the max_delay action were not long enough to include both the boil_water and pour_water actions. The dummy action's duration relies on the duration of the other actions where an "envelope" is used, whereas it does not in the former case. With the "clip" method, the planning and scheduling do not interact in a complex manner (loose coupling), whereas in the second they do interact (tight coupling). Could it be possible to detect such cases with envelopes and translate the problem to use the easier "clipping" method? In this particular case it would seem so.

Returning to the match domain (Figure 2) it is possible to change it such that there is a delay between fixing the two fuses in order to get the fuse out of your pocket. When this is done, it looks identical in structure to the cases discussed in Section 4, with minimum and maximum delays (see Figure 7). Would it then be possible to translate this domain, as the breakfast domain could be, into a form in which the planning and scheduling do not interact? As in the breakfast domain, this would mean changing the duration and structure of the LIGHT_MATCH action, which would result in a plan where you start to fix the first fuse, light the match, finish fixing the fuse, find the next fuse and start fixing that, when the match would run out, and then you could finish the second fuse. This obviously does not make sense, but this plan could be translated back again in a post processing step to the original form. Whilst this guarantees that there is enough time to fix the fuses, the problem then arises, what if something else relies on the duration and structure of the LIGHT_MATCH action? Clearly, if the LIGHT_MATCH action has changed duration, there would be difficulties with anything else requiring that light. It is unclear how the translation would work in the case where there are three or more fuses to fix. Bearing all this evidence in mind, this is not a viable solution to the problem in this case.

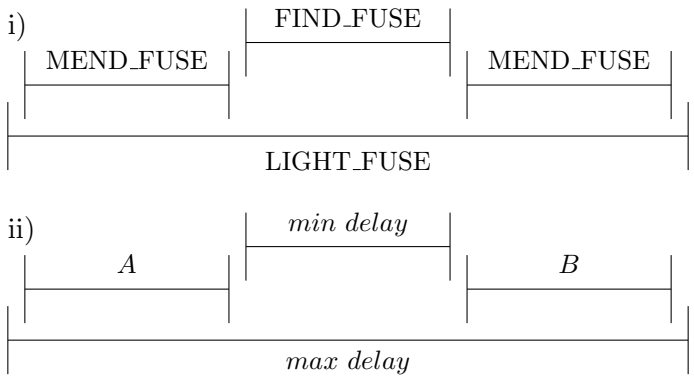

Figure 7: Comparison of the Match Domain and Minimum and Maximum Delays in PDDL2.1

Whilst the representations are syntactically equivalent, they are not always semantically equivalent. The actions that need translating, those which are the equivalent of the dummy maximum and minimum actions, may not actually *be* dummy actions and have other conditions and effects that are important to the domain. Also, if you were actually encoding the temporal constraints set out in Section 4, you could make the assumption that the domain encoder ensures all minimum values are less than or equal to their corresponding maximum values. However, you cannot make the same assumptions about actions that coincidentally are like the temporal constraints. e.g. you cannot assume that FIND_FUSE_DUR $\leq$ LIGHT_MATCH_DUR. A further example could be "the lazy driver", where a driver of a manual transmission car wishes to stop at traffic lights and pull off again but without taking the car out of gear. They must depress the clutch before finishing braking so as not to stall, and then accelerate before taking their foot off the clutch, again, so as to avoid stalling. The plan would be as in Figure 8. This uses a clip, and is syntactically the same as forcing a maximum time. However, as it is semantically different (i.e. *not* forcing a maximum time), any translation into an envelope would change those semantics.

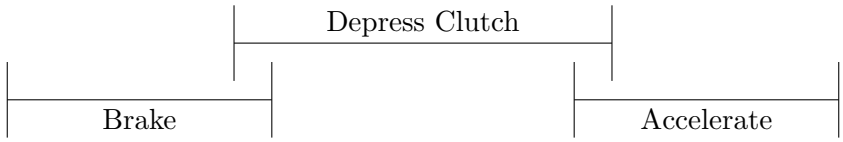

Figure 8: The Lazy Driver

For these reasons, it would seem that in the general case you cannot always loosen the coupling between the planning and scheduling sub-problems through translation of the domain in this manner as the sub-problems interact too much. Whilst this paper only claims that it is impossible separate the problems through the proposed translation, there is good reason to believe that it is impossible to completely separate temporal planning problems into hierarchical sub-problems such that the system described in Section 2 would be complete. Take once more the match domain. If more than one match is needed to fix the fuses, then this decision must be made during the planning stage[3] although it is enforced by the scheduling sub-problem. This should make it clear how the

[3]In fact, one solution could be to post process an un-schedulable plan to make it schedulable by adding more

two problems are tightly coupled together. The next section looks at the scheduling constraints that interact with the planning sub-problem. If these can be detected it could still be possible to separate the problems where possible (which, in most domains, will probably be in the majority of cases) and so gain the advantages of that approach. But more than that, it may be possible to separate as much of the temporal planning problem as possible, and only perform the planning and scheduling together where necessary.

6 Detecting Cases and an Initial Proposed Solution

Cases where the temporal reasoning and logical reasoning need to be detected could either be done whilst the planning is taking place, or by analysing the domain before planning. They occur where you have a sequence of actions that must complete before another set of actions do. It can be seen as an envelope (the bounding actions) which can only hold a certain amount of content (the enclosed actions). If you try to put too much in the envelope, then an un-schedulable plan is produced. The envelope is produced by a sequence of actions which conform to the maximum temporal constraints (as in Figure 4), and the contents, by a sequence of actions whose orderings are one of the minimal temporal constraints (again, as in Figure 4). Should either of these sequences be broken by an ordering of the other type, then either the envelope is broken and it has an infinite capacity, or the contents can be folded up infinitely. In either case, the problems no longer interact. The bounds of either the envelope or the contents can be from any of the start or end points in the constraints. Figure 9 shows possible scenarios. The first is where a single action forms an envelope, with three content actions that must fit in it the duration of the envelope. The second shows how more than one action could form an envelope. The third shows a possible complex case where a sequence of actions with their precedence relationships form both an envelope and contents. There could also be recursive cases, where there are envelopes within envelopes, or contents within contents.

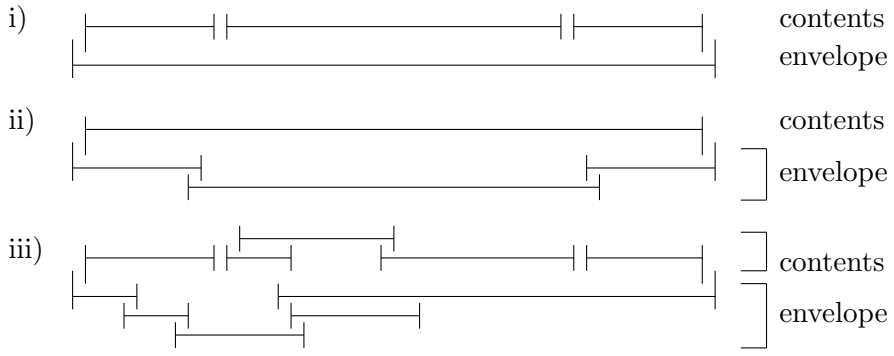

Figure 9: Envelopes and Contents

It is intended to implement a forward heuristic planner that takes this approach of separating scheduling and planning, integrating them only where necessary. Forward heuristic planners have proved to be successful (for example FF [10]), however, these present a particular problem in this case, since once an envelope is started you cannot backtrack over that decision (or at least, the

LIGHT_MATCH actions, but this idea is not developed here. Besides, this could be seen as part of the 'planning' stage.

current version of FF does not backtrack until it starts Breadth First Search). A first step will be taken by pre-processing and looking for the simplest of cases: where an invariant must be met which is provided only during another action (as in the match domain). As forward planning progresses, should such an action be chosen, the algorithm will make sure there is enough time to complete the content actions with the invariants, thus producing a schedulable plan.

In this way, time will be treated as a resource. However this could be a mistake, since it is different in a few key aspects. Firstly, time proceeds forwards at the same rate regardless; no action can either start or stop its consumption or production as other resources such as fuel. Secondly, often it is not the absolute value of time that is of importance, but rather its relative value. For example, it is important to know how much time has elapsed between two events, but for fuel, it is more important to know how much is in the tank. Finally, and most significantly, where actions happen concurrently it is a resource which can be used infinitely at the same "time", without changing its value. This is in contrast against, for example, a processor where only one job can be performed at once, or a tank of fuel, where the more generators that use it, the quicker it runs down. For this reason, whilst initial experiments have been successful, if the fuse action did not require a hand free (i.e. both fuses could be fixed concurrently) only one match would be needed. This can be remedied by having each fuse require its own "time" resource.

7 Final Remarks

This is still work in progress. To briefly recap, planning is easier when the planning and scheduling parts do not interact so that one half of the problem can be ignored whilst the other half is solved. However, in cases where they do interact, the problems need to be solved together. This paper has reviewed where the logical structure of a plan and the temporal constraints interact so that they cannot be solved independently. Most temporal planners always do this reasoning together, however, it would be advantageous only to solve both simultaneously where necessary. In the architecture described in Section 2, knowing where the problems interact can be used to focus search in these areas, and also to see dead ends in the planning space before getting to them. i.e. it is better to know that there won't be time to complete a sequence of actions when you put in the first action, rather than finding this out once the last action is put in. With this combined reasoning, the planner will then not produce un-schedulable plans.

References

[1] R. Dechter, J. Meiri, and J. Pearl. Temporal constraint networks. In *Proceedings from Principles of Knowledge Representation and Reasoning*, pages 83–93. Toronto, Canada, 1989.

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

[3] M. Fox and D. Long. HybridSTAN: Identifying and managing combinatorial optimisation sub- problems in planning. In *Proceedings of the 12th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 445–452, 2001.

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

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

[6] M. Fox, D. Long, and M. Hamdi. Handling multiple sub-problems within a planning domain. In *Proceedings of the 20th UK Planning and Scheduling Special Interest Group (PlanSIG)*, 2001.

[7] A. Garrido, M. Fox, and D. Long. A temporal planning system to manage level 3 durative actions of PDDL2.1. In *Proceedings of the 20th UK Planning and Scheduling Special Interest Group (PlanSIG)*, pages 127–138, 2001.

[8] A. Gerevini and I Serina. LPG: A planner based on local search for planning graphs. In *Proceedings of the 6th International Conference of Artificial Intelligence Planning and Scheduling (AIPS'02)*. AAAI Press, 2002.

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

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

[11] B. Srivastava. Realplan: Decoupling causal and resource reasoning in planning. In *Proceedings from the Twelfth Innovative Applications of Artificial Intelligence Conference on Artificial Intelligence (IAAI)*, pages 812–818, 2000.