Sapa: A Scalable Multi-objective Heuristic Metric Temporal Planner

Minh B. Do, Subbarao Kambhampati [*]
Department of Computer Science and Engineering
Arizona State University, Tempe AZ 85287-5406
{binhminh,rao}@asu.edu

Abstract

In this research paper, we discuss *Sapa*, a domain-independent heuristic forward chaining planner that can handle durative actions, metric resource constraints, and deadline goals. It is designed to be capable of handling the multi-objective nature of the metric temporal planning. Our technical contributions include discussion of (i) various objective functions for planning and the multi-objective search framework (ii) a planning-graph based method for deriving heuristics that are sensitive to both cost and makespan (iii) an easy way of adjusting the heuristic estimates to take the metric resource limitations into account and (iv) a linear time greedy post-processing technique to improve the solution's execution flexibility and quality according to given criteria. An implementation of *Sapa* using a subset of the techniques presented in this paper was one of the best domain independent planners for domains with metric and temporal constraints in the third International Planning Competition, held at AIPS-02. We describe the technical details of extracting the heuristics and present an empirical evaluation of the current implementation of *Sapa*.

1 Introduction

The success of the Deep Space Remote Agent experiment has demonstrated the promise and importance of scalable metric temporal planning for NASA applications. HSTS/RAX, the planner used in the remote agent experiment, was predicated on the availability of domain- and planner-dependent control knowledge, the collection and maintenance of which is admittedly a laborious and error-prone activity. An obvious question of course is whether it will be possible to develop *domain-independent* metric temporal planners that are capable of scaling up to such domains. The past experience has not particularly been encouraging. Although there have been some ambitious attempts–including IxTeT [18] and Zeno [32], their performance has not been particularly satisfactory.

Some encouraging signs however are the recent successes of domain-independent heuristic planning techniques in classical planning (c.f. AltAlt [29] HSP [4] and FF [20]). Our research is

[*]We thank Daniel Bryce for developing the GUI for *Sapa*. This research is supported in part by the NSF grant IRI-9801676, and the NASA grants NAG2-1461 and NCC-1225.

aimed at building on these successes to develop a scalable metric temporal planner. At first blush search control for metric temporal planners would seem to be a very simple matter of adapting the work in heuristic planners in classical planning [4, 29, 20]. The adaptation however does pose several challenges:

- Metric temporal planners tend to have significantly larger search spaces than classical planners. After all, the problem of planning in the presence of durative actions and metric resources subsumes both the classical planning and scheduling problems.
- Compared to classical planners, which only have to handle the logical constraints between actions, metric temporal planners have to deal with many additional types of constraints that involve time and continuous functions representing different aspects of resource.
- In contrast to classical planning, where the only objective is to find shortest length plans, metric temporal planning is *multi-objective*. The user may be interested in improving either temporal quality of the plan (e.g. makespan) or its cost (e.g. cumulative action cost, cost of resources consumed etc.), or more generally, a combination there of. Consequently, effective plan synthesis requires heuristics that are able to track both these aspects in an evolving plan. Things are further complicated by the fact that these aspects are often inter-dependent. For example, it is often possible to find a "cheaper" plan for achieving goals, if we are allowed more time to achieve them.

In this paper, we present *Sapa*, a heuristic metric temporal planner that we are currently developing, to overcome these challenges. *Sapa* is a forward chaining metric temporal planner, which searches in the space of time-stamped states (see below [1]). *Sapa* handles durative actions as well as actions consuming continuous resources. Our main focus has been on the development of heuristics for focusing *Sapa*'s multi-objective search. These heuristics are derived from the optimistic reachability information encoded in the planning graphs. Unlike classical planning heuristics (c.f. AltAlt [29]), which need only estimate the "length" of the plan needed to achieve a set of goals, *Sapa*'s heuristics need to be sensitive to both the cost and length ("makespan") of the plans for achieving the goals. Our contributions include:

- We present a novel framework for tracking the cost of literals (goals) as a function of time. These "cost functions" are then used to derive heuristics that are capable of driving the search towards plans that satisfy any type of cost-makespan tradeoffs.
- *Sapa* generalizes the notion of "phased" relaxation used in deriving heuristics in planners such as AltAlt and FF [29, 20]. Specifically, the heuristics are first derived from a relaxation that ignores the delete effects and metric resource constraints, and are then adjusted subsequently to better account for both negative interactions and resource constraints.
- While *Sapa*'s forward-chaining search in the time-stamped states results in position-constrained plans, it improves the temporal flexibility of the solution plans by post-processing the position-constrained plans to produce equivalent "precedence-constrained" plans. This way, *Sapa* is able to exploit both the ease of resource reasoning offered by the position-constrained plans and the execution flexibility offered by the precedence-constrained plans.

We discuss and implement the linear time greedy approach to generate an o.c plan of better or equal makespan value compare to the original p.c plan. This approach can be thought of as a specific variable and value ordering for the CSOP encoding discussed in [10].

A version of *Sapa* using a subset of the techniques discussed in this paper was one of the best domain independent planners for domains with metric and temporal constraints in the third International Planning Competition, held at AIPS 2002[13]. In fact, it is the best planner in terms of solution quality and number of problems solved in the highest level of PDDL2.1 used in the competition for the domains *Satellite* and *Rovers*. These domains are both inspired by NASA applications.

The paper is organized as follows: in Section 2 we discuss the forward state space search algorithm to produce concurrent metric temporal plans with durative actions. In that section, Section 2.1 is used to describe the action representation and constraints; and Section 2.2 concentrates on the forward chaining algorithm. Next, in Section 3, we discuss the problem of how to build a temporal planning graph and use it to propagate the time and cost information (time-sensitive cost functions). Section 4 shows how the propagated information can be used to estimate the cost of achieving the goals from a given state. We also discuss in that section how the ignored mutual exclusion relations and resource information help improve the heuristic estimation. To improve the quality of the solution, Section 5 discusses our greedy approach of building the order constrained plan (aka. partial order plan) from parallel position constrained plans returned by *Sapa*. Section 6 discusses the implementation of *Sapa*, presents some empirical results of *Sapa* in producing plans with tradeoff for cost and makespan, and analyzes its performance in the international planning competition. We conclude the paper with a discussion on related work in Section 7 and on the conclusion in Section 8.

2 Handling concurrent actions in a forward state space planner

Sapa addresses planning problems that involve durative actions, metric resources, and deadline goals. In this section, we describe how such planning problems are represented and solved in *Sapa*. We first describe the action representation, and will then present the forward chaining state search algorithm used by *Sapa*.

2.1 Action Representation & Constraints

Planning is the problem of finding a set of actions and their respective execution times to satisfy all causal, metric, and resource constraints. Therefore, action representation has influences on the representation of the plans and on the planning algorithm. In this section, we will briefly describe the PDDL2.1 Level 3 action representation, which is the highest level (in terms of the expressiveness of temporal and metric resource constraints) used in the third international competition. *Sapa* is able to solve problems in PDDL2.1 Level 3 and was one of the best of such planners in the competition.[1]

[1]The original action representation used in Sapa is slightly different from the PDDL2.1 Level 3. However, we will commit to the standard action representation from now on.

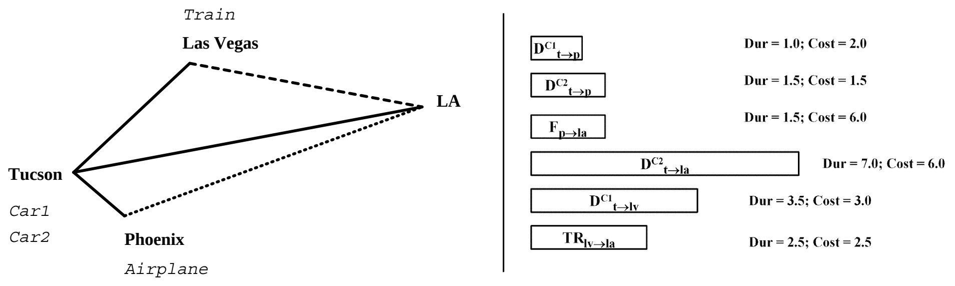


Figure 1: The travel example

Unlike actions in classical planning, in planning problems with temporal and resource constraints, actions are not instantaneous but have durations. Each action *A* has a duration D_A, starting time S_A, and end time ($E_A = S_A + D_A$). The value of D_A can be statically defined for a domain, statically defined for a particular planning problem, or can be dynamically decided at the time of execution.[2] An action *A* can have preconditions $Pre(A)$ that may be required either to be instantaneously true at the time point S_A, or required to be true starting at S_A and remain true for some duration $d \leq D_A$. The logical effects *Eff(A)* of *A* will be divided into two sets $E_s(A)$, and $E_e(A)$ containing, respectively, the instantaneous effects at time points S_A, and E_A.

Actions can also consume or produce metric resources and their preconditions may also depend on the value of the corresponding resource. For resource related preconditions, we allow several types of equality or inequality checking including ==, <, >, <=, >=. For resource-related effects, we allow the following types of change (update): assignment(=), increment(+=), decrement(-=), multiplication(*=), and division(/=).

We shall now illustrate the action representation in a simple temporal planning problem. This problem will be used as the running example through out the rest of the paper. Figure 1 shows graphically the problem description. In this problem, a group of students in Tucson need to go to Los Angeles (LA). Between two options of renting a car, if the students rent a faster but more expensive car (*Car1*), they can only go to Phoenix (PHX) or Las Vegas (LV). However, if they decide to rent a slower but cheaper *Car2*, then they can use it to drive to Phoenix or directly to LA. Moreover, to reach LA, the students can also take a *train* from LV or a flight from PHX. In total, there are 6 movement actions in the domain: *drive-car1-tucson-phoenix* ($D^{c1}_{t\to p}$, *Dur = 1.0, Cost = 2.0*), *drive-car1-tucson-lv* ($D^{c1}_{t\to lv}$, *Dur = 3.5, Cost = 3.0*), *drive-car2-tucson-phoenix* ($D^{c2}_{t\to p}$, *Dur = 1.5, Cost = 1.5*), *drive-car2-tucson-la* ($D^{c2}_{t\to la}$),*Dur = 7.0, Cost = 6.0*, *fly-airplane-phoenix-la* ($F_{p\to la}$, *Dur = 1.5, Cost = 6.0*), and *use-train-lv-la* ($T_{lv\to la}$, *Dur = 2.5, Cost = 2.5*). Each move (by car/airplane/train) action A between two cities X and Y requires the precondition that the students be at X ($at(X)$)at the beginning of A. There are

[2]For example, in the traveling domain discussed in this section, we can decide that boarding a passenger always takes 10 minutes for all problems in this domain. Duration of the action of flying an airplane between two cities will depend on the distance between these two cities and the speed of the airplane. Because the distance between two cities will not change over time, the duration of a particular flying action will be totally specified once we parse the planning problem. However, *refueling* an airplane may have a duration that depends on the current fuel level of that airplane. We may only be able to calculate the duration of a given *refueling* action according to the fuel level at the exact time instant when we execute that action.

also two temporal effects: $\neg at(X)$ occurs at the starting time point of A and $at(Y)$ at the end time point of A. Driving and flying actions also consume different types of resource (e.g fuel) at different rates depending on the specific car or airplane used. In addition, there are refueling actions for cars and airplanes. The durations of the refueling actions depend on the amount of fuel remaining in the vehicle and the refueling rate. The summaries of action specifications for this example are shown on the right side of Figure 1.

In this example, the costs of moving by train or airplane are the respective ticket prices, and the costs of moving by rental cars include the rental fees and gas (resource) costs.

2.2 A Forward Chaining Search Algorithm for metric temporal planning

Even though variations of the action representation scheme described in the previous section have been used in the partial order temporal planners such as IxTeT[18] and Zeno[32] before, Bacchus and Ady [1] are the first to propose a forward chaining algorithm capable of using this type of action representation allowing concurrent execution of actions in the plan. We adapt their search algorithm in *Sapa*.

Sapa's search is conducted through the space of time stamped states. We define a time stamped state S as a tuple $S = (P, M, \Pi, Q, t)$ consisting of the following structure:

- $P = (\langle p_i, t_i \rangle \mid t_i < t)$ is a set of predicates p_i that are true at *t* and the last time instant t_i at which they are achieved.
- *M* is a set of values for all continuous functions, which may change over the course of planning. Functions are used to represent the metric-resources and other continuous values. Examples of functions are the fuel levels of vehicles.
- Π is a set of persistent conditions, such as durative preconditions, that need to be protected during a specific period of time.
- *Q* is an event queue containing a set of updates each scheduled to occur at a specified time in the future. An event *e* can do one of three things: (1) change the True/False value of some predicate, (2) update the value of some function representing a metric-resource, or (3) end the persistence of some condition.
- t is the time stamp of S

In this paper, unless noted otherwise, when we say “state” we mean a time stamped state. It should be obvious that time stamped states do not just describe world states (or snap shots of the world at a given point of time) as done in classical progression planners, but rather describe both the current state of the world and the state of the planner's search (set of events/changes that are guaranteed to occur given the actions in the current plan).

The initial state S_{init} is stamped at time 0 and has an empty event queue and empty set of persistent conditions. It is completely specified in terms of function and predicate values. The goals are represented by a set of 2-tuples $G = (\langle p_1, t_1 \rangle ... \langle p_n, t_n \rangle)$ where p_i is the i^{th} goal and t_i is the time instant by which p_i needs to be achieved. Note that PDDL2.1 still does not allow the specification of goal deadline constraints. Therefore, we currently use our original representation for problems requiring goal deadline constraints. We intend to extend PDDL2.1 in the near future to include goal deadlines and other types of temporal constraints.

Goal Satisfaction: The state $S = (P, M, \Pi, Q, t)$ *subsumes* (entails) the goal *G* if for each $\langle p_i, t_i \rangle \in G$ either:

1. $\exists \langle p_i, t_j \rangle \in P$, $t_j < t_i$ and there is no event in *Q* that deletes p_i.

2. $\exists e \in Q$ that adds p_i at time instant $t_e < t_i$.

Action Application: An action A is *applicable* in state $S = (P, M, \Pi, Q, t)$ if:

1. All logical (pre)conditions of *A* are satisfied by *P*.

2. All metric resource conditions of A are satisfied by *M*.[3]

3. *A*'s effects do not interfere with any persistent condition in Π and any event in *Q*.

4. $\nexists e \in Q$ interferes with persistent preconditions of *A*.

The *interference* relation between action A's effects, event queue Q, and the persistent condition Π follow the interference rules defined in PDDL2.1. In short, they are:

- If A deletes a proposition p and there is an event e in Q that adds p, then A can not delete P at the same time instant at which that event occurs (epsilon or higher duration should separate two events).

- If A deletes p and p is protected in Π until time point t_p, then A should not delete p before t_p.

- If A has a persistent precondition p, and there is an event that gives $\neg p$, then that event should occur *after* A terminates.

- A should not change the value of any function (representing a resource) which is used to satisfy a (pre)condition of another unterminated action[4]. Moreover, if A has a precondition that depends on the value of some function that is changed as an effect of an unterminated action A', then A also interferes with that action (and is not applicable at the current state).

When we apply an action *A* to a state $S = (P, M, \Pi, Q, t)$, all instantaneous effects of *A* will be immediately used to update the predicate list *P* and metric resources database *M* of *S*. *A*'s persistent preconditions and delayed effects will be put into the persistent condition set Π and event queue *Q* of *S*.

[3]For example, if the condition to execute an action $A = move(truck, A, B)$ is $fuel(truck) > 500$ then A is executable in S if the value v of $fuel(truck)$ in M satisfy $v > 500$.

[4]Unterminated actions are the ones that started before the time point t of the current state S but have not yet finished at t.

```
State Queue: $SQ$={S$_{init}$}
while $SQ \neq \{\}$
    $S$:= Dequeue($SQ$)
    Nondeterministically select $A$ applicable in $S$
        $S'$ := Apply($A$,$S$)
        if $S'$ satisfies $G$ then PrintSolution
        else Enqueue($S'$,$SQ$)
end while;
```

Figure 2: Main search algorithm

Besides the normal actions, we will have a special action called **advance-time**[5] which we use to advance the time stamp of *S* to the time instant t_e of the earliest event e in the event queue *Q* of *S*. The advance-time action will be applicable in any state *S* that has a non-empty event queue. Upon applying this action, the state *S* gets updated according to all the events in the event queue that are scheduled to occur at t_e.
Search algorithm: The basic algorithm for searching in the space of time stamped states is shown in Figure 2. We proceed by applying all applicable actions to the current state and put the result states into the sorted queue using the $Enqueue()$ function. The $Dequeue()$ function is used to take out the first state from the state queue. Currently, *Sapa* employs the A* search. Thus, the state queue is sorted according to some heuristic function that measures the difficulty of reaching the goals from the current state. The rest of the paper discusses the design of such heuristic functions.

3 Propagating time-sensitive cost functions using the temporal planning graphs

In this section, we discuss the issue of deriving heuristics, that are sensitive to both time and cost, to guide *Sapa*'s search algorithm. An important challenge in finding heuristics to support multi-objective search, as illustrated by the example below, is that the cost and temporal aspects of a plan are often inter-dependent. Therefore, in this section, we introduce the approach to track the cost of achieving goals and execute actions in the plan as the functions of time. Then, the propagated cost functions can be used to derive the heuristic values to guide the search in *Sapa*.
Example: Let's take a simpler version of our ongoing example. Suppose that we need to go from Tucson to Los Angeles. The two common options are: (i) rent a car and drive from Tucson to Los Angeles in one day for \$100 or (ii) take a shuttle to the Phoenix airport and fly to Los Angeles in 3 hours for \$200. The first option takes more time (higher makespan) but less money, while the second one clearly takes less time but is more expensive. Depending on the specific weights the user gives to each criterion, she may prefer the first option over the second or vice versa. Moreover, the user's decision may also be influenced by other constraints on time and cost that are imposed over the final plan. For example, if she needs to be in Los Angeles in six

[5]*Advance-time* is called **unqueue-event** in [1]

hours, then she may be forced to choose the second option. However, if she has plenty of time but limited budget, then she may choose the first option.

The simple example above shows that makespan and execution cost, while nominally independent of each other, are nevertheless related in terms of the overall objectives of the user and the constraints on a given planning problem. More specifically, for a given makespan threshold (e.g. to be in LA within six hours), there is a certain estimated solution cost tied to it (shuttle fee and ticket price to LA) and vice versa. Thus, in order to find plans that are good with respect to both cost and makespan, we need to develop heuristics that track cost of a set of (sub)goals as a function of time.

Given that the planning graph is an excellent structure to represent the relation between facts and actions, we will use the temporal planning graph structure (TGP[35]) as a substrate for propagating the cost information. For the rest of this section, we start with a brief discussion of the data structures used for the cost propagation process in Section 3.1. We then continue with the details of the propagation process in Section 3.2, and the criteria used to terminate the propagation in Section 3.3.

3.1 The Temporal Planning Graph Structure

We now adapt the notion of temporal planning graphs, introduced in [35] to our action representation. The temporal planning graph for a given problem is a bi-level graph, with one level containing all *facts*, and the other containing all *actions* in the planning problem. Each fact links to all actions supporting it, and each action links to all facts that belong to its precondition and effect lists.[6] Actions are durative and their effects are represented as events that occur at some time between the action's start and end time points. As we will see in more detail in the later parts of this section, we build the temporal planning graph by incrementally increasing the time (makespan value) of the graph. At a given time point t, an action A is activated if all preconditions of A can be achieved at t. To support the *delayed effects* of the activated actions (i.e., effects that occur at the *future* time points beyond t), for the whole graph, we also maintain an event queue $Q = \{e_1, e_2, ...e_n\}$ sorted in the increasing order of event time. The event queue discussed in this section is slightly different from the one discussed in the previous section. In short, the event queue:

- Is associated with the whole planning graph (rather than single actions).
- Only contains the *positive* events (add predicate)
- Has an *event cost* associated with it.

Each event is a 4-tuple $e = \langle f, t, c, A \rangle$ in which: (1) f is the fact that e will add; (2) t is the time point at which the event will occur; and (3) c is the cost incurred to enable the execution of action A which causes e. For each action A, we introduce a cost function $C(A, t) = v$ to

[6]The bi-level representation has been used in the classical planning to save time and space, but as Smith & Weld[35] showed, it makes even more sense in the temporal planning domains because there is actually no notion of level. All we have are a set of facts/action nodes, each one encoding information such as the *earliest time point* at which the fact/action can be achieved/executed, and the *lowest cost* incurred to achieve them etc.

specify the estimated cost v that we incur to enable A's execution at time point t. In other words, $C(A, t)$ is the estimate of the cost incurred to achieve all of A's preconditions at time point t. Moreover, each action will also have an *execution cost* ($C_{exec}(A)$), which is the cost incurred in executing A (e.g ticket price for *fly* action, gas cost for driving a car). For each fact f, a similar cost function $C(f, t) = v$ specifies the estimated cost v incurred to achieve f at time point t. We also need one additional function $SA(f, t) = A_f$ to specify the action A_f that can be used to *support* f with cost v at time point t. Due to the fact that actions have different durations, we will show later that the values of the cost functions $C(A, t), C(f, t)$ monotonically decrease over time. The temporal planning graph of the ongoing example, with each action is at its earliest starting time, is shown in the left side of Figure 4.

In the original Graphplan algorithm[3], there is a process of propagating mutex information, which captures the negative interactions between different propositions and actions occurring at the same time (level). To simplify the discussion, in this paper we will neglect the mutex propagation and will discuss the propagation process in the context of the relaxed problem in which the *delete effects* of actions, which cause the mutex relations, are ignored. In Section 4.2.1, we will discuss how the temporal mutex relations such as the ones discussed in TGP[35] can be used to improve the cost propagation and heuristic estimation processes.

3.2 Cost propagation procedure

As mentioned above, our general approach is to propagate the estimated costs incurred to achieve facts and actions from the initial state. As a first step, we need to initialize the cost functions $C(A, t)$ and $C(f, t)$ for all facts and actions. For a given initial state S_{init}, let $F = \{f_1, f_2 ... f_n\}$ be the set of facts that are true at time point t_{init} and $\{(f'_1, t_1), ...(f'_m, t_m)\}$, [7] be a set of outstanding positive events which specify the addition of facts f'_i at time points $t_i > t_{init}$. We introduce a dummy action A_{init} to represent S_{init} where A_{init} (i) requires no preconditions; (ii) has cost $C_{exec}(A_{init}) = 0$ and (iii) causes the events of adding all f_i at t_{init} and f'_i at time points t_i. At the beginning ($t = 0$), the event queue Q is empty, the cost functions for all facts and actions are initialized as: $C(A, t) = \infty, C(f, t) = \infty,\ \forall 0 \leq t < \infty$, and A_{init} is the only action that is applicable.

Figure 3 summarizes the steps in the cost propagation algorithm. The main algorithm contains two interleaved parts: one for applying an action and the other for activating an event representing the action's effect.

Applying an action: When an action A is applied, we (1) augment the event queue Q with events corresponding to all of A's effects, and (2) update the cost function $C(A, t)$ of A.

Activating an event: When an event $e = \langle f_e, t_e, C_e, A_e \rangle$, which represents an effect of A_e occurring at time point t_e and adding a fact f_e with cost C_e is activated, the cost function of the fact f_e is updated if $C_e < C(f_e, t_e)$. Moreover, if the newly improved cost of f_e leads to a reduction in the cost functions of an action A that f_e supports (as decided by function $CostAggregate(A, t)$ in line 11 of Figure 3) then we will *(re)apply* A to propagate f_e's new cost of achievement to the cost functions of A and its effects.

[7] In temporal planning, because actions may have non-uniform duration, at any given time point, the state may not only consist of facts which are true, but also events representing delayed effects of a selected action that occurs in the future.

```
**Function** *Propagate Cost*
  Current time: $t_c = 0$;
  $Apply(A_{init}, 0)$;
  **while** *Termination-Criteria* $\neq$ *true*
    Get earliest event $e = \langle f_e, t_e, c_e, A_e \rangle$ from $Q$;
    $t_c = t_e$;
    **if** $c_e < C(f, t_c)$ **then**
    Update: $C(f, t) = c_e$
      **for all** action $A$: $f \in Precondition(A)$
        $NewCost_A = CostAggregate(A, t_c)$;
        **if** $NewCost_A < C(A, t_c)$ **then**
          Update: $C(A, t) = NewCost(A), t_c \leq t < \infty$;
          $Apply(A, t_c)$;
**End** *Propagate Cost*;

**Function** $Apply(A, t)$
   **For all** A's effect that add $f$ at $S_A + d$ **do**
         $Q = Q \bigcup \{e = \langle f, t + d, C(A, t) + C_{exec}(A), A \rangle\}$;
**End** $Apply(A, t)$;
```

Figure 3: Main cost propagation algorithm

At any given time point t, $C(A, t)$ is an aggregated cost (returned by function $CostAggregate(A, t)$) to achieve all of its preconditions. The aggregation can be done in different ways:

1. **Max-propagation:**
   $C(A, t) = Max\{C(f, t) : f \in Precond(A)\}$ or

2. **Sum-propagation:**
   $C(A, t) = \sum\{C(f, t) : f \in Precond(A)\}$ or

3. **Combo:**
   $C(A, t) = 0.5 * (Max\{C(f, t) : f \in Precond(A)\}) + 0.5 * (\sum\{C(f, t) : f \in Precond(A)\})$

The first method assumes that all preconditions of an action depend on each other and the cost to achieve all of them is equal to the cost to achieve the costliest one. This rule leads to the underestimation of $C(A, t)$ and the value of $C(A, t)$ is admissible. The second method (*sum-propagation*) assumes that all facts are independent. Although clearly inadmissible, it has been shown (c.f.[29, 4]) to be more effective than the *max-propagation* for the case of makespan propagation. The last method combines the two and basically tries to account for the dependency between different facts in the *sum-propagation*.

When the cost function of one of the preconditions of a given action is updated (lowered), the $CostAggregate(A, t)$ function is called and it uses one of the methods described above

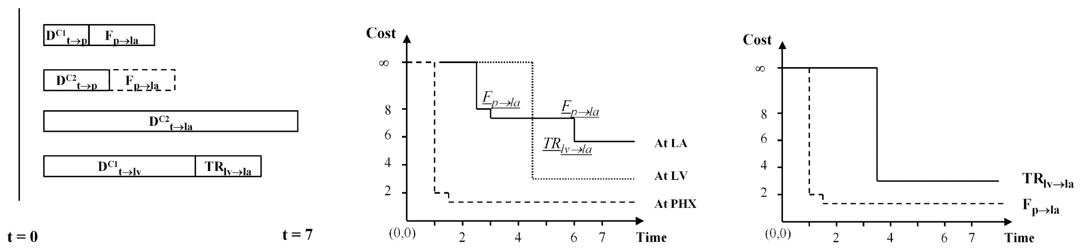


Figure 4: Cost functions for facts and actions in the travel example.

to calculate if the cost required to execute an action has improved (reduced).[8] If $C(A,t)$ has improved, then we will *re-apply* A (line 12-14 in Figure 3) to propagate the improved cost $C(A,t)$ to the cost functions $C(f,t)$ of its effects.

Notice that the way we update the cost functions of facts and actions in the planning domains described above shows the challenges in heuristic estimation in temporal planning domains. Because action's effects do not occur instantaneously at the action's starting time, concurrent actions overlap in many possible ways and thus the cost functions, which represent the difficulty of achieving facts and actions are *time-sensitive*. Finally, the only remaining issue in the main algorithm illustrated in Figure 3 is the *termination criteria* for the propagation, which will be discussed in detail in Section 3.3.

Before demonstrating the cost propagation process in our ongoing example, following are some observations of our propagated cost function discussed in this section:

Observation 1: *The propagated cost functions of facts and actions* are non-increasing *over time.*

Observation 2: *Because we increase time in* step *jump by going through events in the event queue, the cost functions for all facts and actions will appear as* step-functions, *even though time is measured continuously.*

Due to the first observation, the estimated cheapest cost of achieving a given goal g at time point t_g is $C(g,t_g)$. Thus, we do not need to look at the value of $C(g,t)$ at time point $t < t_g$. Moreover, to evaluate the heuristic value for an objective function f involving both time and cost, the second observation helps us to concentrate on computing f at only time points at which the cost function of some fact or action changes. We will comeback to the details of the heuristic estimation routines in the next section (Section 4).

Coming back to our running example, the left side of Figure 4 shows graphically the time points at which each action can be applied ($C(A,t) < \infty$) and the right side shows how the cost function of facts/actions change as the time increases. Here is an outline of the update process in this example: at time point $t = 0$, four actions can be applied. They are $D^{c1}_{t\to p}$, $D^{c2}_{t\to p}$, $D^{c1}_{t\to lv}$, $D^{c2}_{t\to la}$. These actions add 4 events into the event queue $Q = \{e_1 = \langle at\_phx, t = 1.0, c = 2.0, D^{c1}_{t\to p}\rangle, e_2 = \langle at\_phx, 1.5, 1.5, D^{c2}_{t\to p}\rangle, e_3 = \langle at\_lv, 3.5, 3.0, D^{c1}_{t\to lv}\rangle, e_4 = \langle at\_la, 7.0, 6.0, D^{c2}_{t\to la}\rangle\}$. After we advance the time to $t = 1.0$, the first event e_1 is activated and $C(at\_phx, t)$ is updated. Moreover, because at_phx is a precondition of $F_{p\to la}$, we

[8]Propagation rule (2) and (3) will improve (lower) the value of $C(A,t)$ when the cost function of one of A's preconditions is improved. However, for rule (1), the value of $C(A,t)$ is improved only when the cost function of its costliest precondition is updated.

also update $C(F_{p\rightarrow la}, t)$ at $t_e = 1.0$ from ∞ to 2.0 and put an event $e = \langle at\_la, 2.5, 8.0, F_{p\rightarrow la}\rangle$, which represents $F_{p\rightarrow la}$'s effect, into Q. We then go on with the second event $\langle at\_phx, 1.5, 1.5, D^{c2}_{t\rightarrow p}\rangle$ and lower the cost of the fact at_phx and action $F_{p\rightarrow la}$. Event $e = \langle at\_la, 3.0, 7.5, F_{p\rightarrow la}\rangle$ is added as a result of the newly improved cost of $F_{p\rightarrow la}$. Continuing the process, we update the cost function of at_la once at time point $t = 2.5$, and again at $t = 3.0$ as the delayed effects of actions $F_{p\rightarrow la}$ occur. At time point $t = 3.5$, we update the cost value of at_lv and action $T_{lv\rightarrow la}$ and introduce the event $e = \langle at\_la, 6.0, 5.5, T_{lv\rightarrow la}\rangle$. Notice that the final event $e' = \langle at\_la, 7.0, 6.0, D^{c2}_{t\rightarrow la}\rangle$ representing a delayed effect of action $D^{c2}_{t\rightarrow la}$ applied at $t = 0$ will not cause any update. This is because the cost function of at_la has been updated to value $c = 5.5 < c_{e'}$ at time $t = 6.0 < t_{e'} = 7.0$.

Besides the values of the cost functions, Figure 4 also shows the supporting actions ($SA(f, t)$) for the fact (goal) at_la. We can see that action $T_{lv\rightarrow la}$ gives the best cost of $C(at\_la, t) = 5.5$ for $t \geq 6.0$ and action $F_{p\rightarrow la}$ gives best cost $C(at\_la, t) = 7.5$ for $3.0 \leq t < 5.5$ and $C(at\_la, t) = 8.0$ for $2.5 \leq t < 3.0$. The right most graph in Figure 4 shows similar cost functions for the actions in this example. We only show the cost functions of actions $T_{lv\rightarrow la}$ and $F_{p\rightarrow la}$ because the other four actions are already applicable at time point $t_{init} = 0$ and thus their cost functions stabilize at 0.

3.3 Termination criteria for the cost propagation process

In this section, we discuss the issue of when we should terminate the cost propagation process. The first thing to note is that cost propagation is in some ways inherently more complex than makespan propagation. For example, once a set of literals enter the planning graph (and are not mutually exclusive), the estimate of the makespan of the shortest plan for achieving them does not change as we continue to expand the planning graph. In contrast, the estimate of the cost of the cheapest plan for achieving them can change until the planning graph levels off. This is why we need to carefully consider the effect of different criteria for stopping the expansion of the planning graph on the accuracy of the cost estimates. The first intuition is that we should not stop the propagation when there exist top level goals for which the cost of achievement is still infinite (unreached goal). On the other hand, given our objective function of finding the cheapest way to achieve the goals, we need not continue the propagation when there is no chance that we can improve the cost of achieving the goals. From those intuitions, following are several rules that constrain the termination:

Deadline termination: *The propagation should stop at time point t if: (1)* $\forall\ goal\ G : Deadline(G) \leq t$*, or (2)* $\exists\ goal\ G : (Deadline(G) < t) \wedge (C(G, t) = \infty)$.

The first rule governs the hard constraints on the goal deadlines. It implies that we should not propagate beyond the latest goal deadline (because any cost estimation beyond that point is useless), or we can not achieve some goal by its deadline.

With the observation that the propagated costs can change only if we still have some events left in the queue that can possibly change the cost functions of a specific propositions, we have the second general termination rule regarding the propagation:

Fix-point termination: *The propagation should stop when there are no more events that can decrease the cost of any proposition.*

The second rule is a qualification for reaching the fix-point in which there is no gain on the cost function of any fact or action. It is analogous to the idea of growing the planning graph until it *levels-off* in classical planning.

Stopping the propagation according to the two general rules above leads us to the best (lowest value) achievable cost estimation for all propositions given a specific initial state. However, there are several situations in which we may want to stop the propagation process earlier. First, propagation until the fix-point, where there is no gain on the cost function of any fact or action, may be costly.[9] Second, the cost functions of the goals may reach the fix-point long before the full propagation process is terminated according to the general rules discussed above, where the costs of *all* propositions and actions stabilize.

Given the above motivations, we introduce several different criteria to stop the propagation earlier than is entailed by the fix-point computation:

Zero-lookahead approximation: *Stop the propagation at the earliest time point t where all the goals are reachable ($C(G,t) < \infty$).*

One-lookahead approximation: *At the earliest time point t where all the goals are reachable, execute all the remaining events in the event queue and stop the propagation.*

One-lookahead approximation looks ahead one step in the (future) event queues when one path to achieve all the goals under the relaxed assumption is guaranteed and hopes that executing all those events would explicate some cheaper path to achieve all goals.[10]

Zero and one-lookahead are examples of a more general k-lookahead approximation, in which extracting the heuristic value as soon as all the goals are reachable corresponds to *zero-lookahead* and continuing to propagate until the fix-point corresponds to the *infinite (full) lookahead*. The rationale behind the k-lookahead approximation is that when all the goals appear, which is an indication that there exists at least one (relaxed) solution, then we will look ahead one or more steps to see if we can achieve some extra improvement in the cost of achieving the goals (and thus lead to lower cost solution).[11]

Coming back to our travel example, zero-lookahead stops the propagation process at the time point $t = 2.5$ and the goal cost is $C(in\_la, 2.5) = 8.0$. The action chain giving that cost is $\{(D^{c1}_{t\rightarrow p}, F_{p\rightarrow la}\}$. With one-lookahead, we find the lowest cost for achieving the goal in_la is $C(in\_la, 7.0) = 6.0$ and it is given by the action $(D^{c2}_{t\rightarrow la})$. With two-lookahead approximation, the lowest cost for in_la is $C(in\_la, 6.0) = 5.5$ and it is achieved by cost propagation through the action set $\{(D^{c1}_{t\rightarrow lv}, T_{lv\rightarrow la})\}$. In this example, two-lookahead has the same effect as the fix-point propagation (infinite lookahead) if the deadline to achieve in_la is later than $t = 6.0$. If it is earlier, say $Deadline(in\_la) = 5.5$, then the one-lookahead will have the same effect as the infinite-lookahead option and gives the cost of $C(in\_la, 3.0) = 7.5$ for the action chain $\{D^{c2}_{t\rightarrow phx}, F_{phx\rightarrow la}\}$.

[9]It has been pointed out in AltAlt[29] that growing the classical planning graph until it *levels-off* is very costly in many problems.

[10]Note that even if none of those events is directly related to the goals, their executions can still lead to better (cheaper) path to reach all the goals.

[11]For backward planners where we only need to run the propagation one time, infinite-lookahead or higher levels of lookahead may pay off, while in forward planners where we need to evaluate the cost of goals for each single search state, lower values of k may be more appropriate.

4 Heuristics based on propagated cost functions

Once the propagation process terminates, the time-sensitive cost functions contain sufficient information to estimate the heuristic value of any given state. Specifically, suppose the planning graph is grown from a state S. Then the cost functions for the set of goals $G = \{(g_1, t_1), (g_2, t_2)...(g_n, t_n)\}$, $t_i = Deadline(g_i)$ can be used to derive the following estimates:

- The minimum makespan estimate for a plan starting from S, $T(P_S)$ is given by the earliest time point τ_0 at which all goals are reached with finite cost $C(g, t) < \infty$.
- The minimum/maximum/summation estimate of slack(s) for a plan starting from S, $Slack(P_S)$ is given by the minimum/maximum/summation of the distance between the time point at which a goal appears in the temporal planning graph and the deadline of that goal.
- The minimum cost estimate of a plan starting from S and achieving a set of goals G, $C(P_S, \tau_\infty)$, can be computed by aggregating the cost estimates for achieving each of the individual goals at their respective deadlines ($C(g, deadline(g))$).[12] Notice that we use τ_∞ to denote the time point at which the cost propagation process stops. Thus, τ_∞ is the time point at which the cost functions for all individual goals $C(f, \tau_\infty)$ have lowest value.
- For each value $t : \tau_0 < t < \tau_\infty$, the cost estimate of a plan $C(P_S, t)$, which can achieve goals within a given makespan limit of t, is the aggregation of the values $C(g_i, t)$ of goals g_i.

The makespan and the cost estimates of a state can be used as the basis for deriving heuristics. The specific way these estimates are combined to compute the heuristic values does of course depend on what the user's ultimate objective function is. In the general case, the objective would be a function $f(C(P_S), T(P_S))$ involving both the cost ($C(P_S)$) and makespan ($T(P_S)$) values of the plan. Suppose that the objective function is a linear combination of cost and makespan:

$$h(S) = f(C(P_S), T(P_S)) = \alpha.C(P_S) + (1 - \alpha).T(P_S)$$

If the user only cares about the makespan value ($\alpha = 0$), then $h(S) = T(P_S) = \tau_0$. Similarly, if the user only cares about the plan cost ($\alpha = 1$), then $h(S) = C(P_S, \tau_\infty)$. In the more general case, where $0 < \alpha < 1$, then we have to find the time point t, $\tau_0 \leq t \leq \tau_\infty$, such that $h_t(S) = f(C(P_S, t), t) = \alpha.C(P_S, t) + (1 - \alpha).t$ has minimum value.[13]

In our ongoing example, given our goal of being in Los Angeles (*at_la*), if $\alpha = 0$, the heuristic value is $h(S) = \tau_0 = 2.5$ which is the earliest time point at which $C(at\_la, t) < \infty$. The heuristic value corresponds to the propagation through action chain $(D^{c1}_{t \to p}, F_{p \to la})$. If $\alpha =$

[12]If we consider G as the set of preconditions for a dummy action that represents the goal state, then we can use any of the propagation rules (max/sum/combo) discussed in Section 3.2 to directly estimate the total cost of achieving the goals from the given initial state.

[13]Because $f(C(P_S, t), t)$ estimates the cost of the (cheapest) plan that achieves all goals with the makespan value $T(P_S) = t$, the minimum of $f(C(P_S, t)$ ($\tau_0 \leq t \leq \tau_\infty$) estimates the plan P that achieve the goals from state S and P has a smallest value of $f(C(P_S), T(P_S))$. That value would be the heuristic estimation for our objective function of minimizing $f(C(P_S), T(P_S))$.

1 and $Deadline(At_{LA}) \geq 6.0$, then $h(S) = 5.5$, which is the cheapest cost we can get at time point $\tau_\infty = 6.0$. This heuristic value represents another solution $(D^{c1}_{t\rightarrow lv}, T_{lv\rightarrow la})$. Finally, if $0 < \alpha < 1$, say $\alpha = 0.55$, then the lowest heuristic value $h(S) = \alpha.C(P_S, t) + (1-\alpha).t$ is $h(S) = 0.55 * 7.5 + 0.45 * 3.0 = 5.47$ at time point $2.5 < t = 3.0 < 6.0$. For $\alpha = 0.55$, this heuristic value $h(S) = 5.47$ corresponds to yet another solution $(D^{c2}_{t\rightarrow p}, F_{p\rightarrow la})$.

Notice that in the general case where $0 < \alpha < 1$, even though time is measured continuously, we do not need to check every time point t: $\tau_0 < t < \tau_\infty$ to find the value where $h(S) = f(C(P_S,t),t)$ is minimal. This is due to the fact that the cost functions for all facts (including goals) are *step functions*. Thus, we only need to compute $h(S)$ at the time points where one of the cost functions $C(g_i, t)$ changes value. In our example above, we only need to calculate values of $h(S)$ at $\tau_0 = 2.5$, $t = 3.0$ and $\tau_\infty = 6.0$ to realize that $h(S)$ has minimum value at time point $t = 3.0$ for $\alpha = 0.55$.

Before we end this section, we have to note that when there are multiple goals there are several possible ways of computing $C(P_S)$ from the cost functions of the individual goals. This is a consequence of the fact that there are multiple rules to propagate the cost, and there are also interactions between the goals/subgoals. Broadly, there are two different ways to extract the plan costs. We can either directly use the cost functions of the goals to compute $C(P_S)$, or first extract a relaxed plan from the temporal planning graph using the cost functions, and then measure $C(P_S)$ based on the relaxed plan. We discuss these two approaches below.

4.1 Directly using cost functions to estimate $C(P_S)$

After we terminate the propagation using any of the criteria discussed in Section 3.3, let $G = \{(g_1,t_1),(g_2,t_2)...(g_n,t_n)\}$, $t_i = Deadline(g_i)$ be a set of goals and $C_G = \{c_1,...c_n | c_i = C(g_i, Deadline(g_i)\}$ be their best possible achievement costs. If we consider G as the set of preconditions for a dummy action that represents the goal state, then we can use any of the propagation rules (max/sum/combo) discussed in Section 3.2 to directly estimate the total cost of achieving the goals from the given initial state. Among all the different combinations of the propagation rules and the aggregation rules to compute the total cost of the set of goals G, only the *max-max* (max-propagation to update $C(g_i,t)$, and cost of G is the maximum of the values of $C(g_i, Deadline(g_i))$ is admissible. The *sum-sum* rule, which assumes the total independence between all facts, and the other seven combinations are different options to reflect the dependencies between facts in the planning problem. The tradeoffs between them can only be evaluated empirically.

4.2 Computing Cost from the relaxed plan

To take into account the positive interactions between facts in planning problems, we can do a backtrack-free search from the goals to find a relaxed plan. Then, the total execution cost of actions in the relaxed plan and its makespan can be used for the heuristic estimation. Besides providing a possibly better heuristic estimate, work on FF[20] shows that actions in the relaxed plan can also be used, leading to effectively focusing the search on the branches surrounding the relaxed solution. Moreover, extracting the relaxed solution allows us to use the resource adjustment techniques discussed in Section 4.2.2 to improve the heuristic estimations. The challenge

```
Goals: $G = \{(g_1, t_1), (g_2, t_2)...(g_n, t_n)\}$
Actions in the relaxed-plan: $RP = \{\}$
Supported facts: $SF = \{f : f \in InitialStateS\}$
**While** $G \neq \emptyset$
    Select the best action $A$ that support $g_1$
    $RP = RP + A$
    $t_A = t_1 - Dur(A)$
    Update makespan value $T(RP)$ if $t_A < T(RP)$
    **For all** $f \in Effect(A)$ added by $A$ after
        duration $t_f$ from starting point of $A$ **do**
        $SF = SF \bigcup \{(f, t_A + t_f)\}$
    **For all** $f \in Precondition(A)$ s.t $C(f, t_A) > 0$ **do**
        $G = G \bigcup \{(f, t_A)\}$
    **If** $\exists (g_i, t_i) \in G, (g_i, t_j) \in SF : t_j < t_i$ **Then**
        $G = G \setminus \{(g_i, t_i)\}$
**End while;**
```

Figure 5: Procedure to extract the relaxed plan

here is how to use the cost functions to guide the search for the best relaxed plan and we address this below.

The basic idea is to go backward finding actions to achieve the goals. When an action is selected, we add its preconditions to the goal list and remove the goals that are achieved by that action. The partial relaxed plan is the plan containing the selected actions and the causal structure between them. When all the remaining goals are satisfied by the initial state S, we have the complete relaxed plan and the extraction process is finished. At each stage, an action is selected so that the complete relaxed plan that contains the selected actions is likely to have the lowest estimated objective value $f(P_S, T_S)$. For a given initial state S and the objective function $h(S) = f(C(P_S), T(P_S))$, Figure 5 describes a greedy procedure to find a relaxed plan given the temporal planning graph. First, let RP be the set of actions in the relaxed plan, SF be the set of time-stamped facts (f_i, t_i) that are currently supported , and G be the set of current goals. Thus, at the beginning, SF is the collection of facts supported by the initial state S and the effects of actions in RP, and G is the conjunction of top level goals and the set of preconditions of actions in RP that are not currently supported by facts in SF. The estimated heuristic value for the current (partial) relaxed plan and the current goal set is computed as follows: $h(S) = h(RP) + h(G)$ in which $h(RP) = f(C(RP), T(RP))$. For the given set of goals G, $h(G) = min\ f(C(G, t), t) : \tau_0 < t < \tau_\infty$ is calculated according to the approach discussed in the previous section (Section 4). Finally, $C(RP) = \sum_{A \in RP} C_{exec}(A)$ and $T(RP)$ is the makespan of RP, where actions in RP are aligned according to their causal relationship. We will elaborate on this in the example at the later part of this section.

In the beginning, G is the set of top level goals, RP is empty and SF contains facts in the initial state. Thus $C(RP) = 0$, $T(RP) = 0$ and $h(S) = h(G)$. We start the extraction process by going backward searching for the *least expensive* action A supporting the first goal

g_1. By least expensive, we mean that A contributes the smallest amount to the objective function $h(S) = h(RP) + h(G)$ if A is added to the current relaxed plan. Specifically, for each action A that supports g_1, we calculate the value $h_A(S) = h(RP + A) + h((G \setminus g_1) \bigcup Precond(A))$ which estimates the heuristic value if we add A to the relaxed plan. We then choose the action A that has the smallest $h_A(S)$ value. [14]

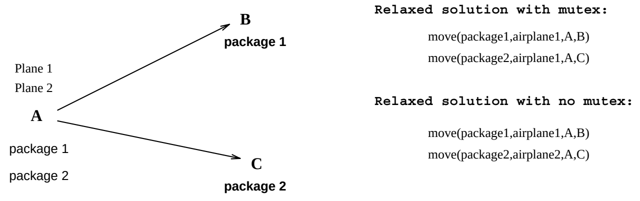
When an action A is chosen, we put its preconditions into the current goal list G, and its effects into the set of supported facts SF. Moreover, we order A to be executed and finished *before* the action that has g_1 as its precondition. Using those ordering relations between actions in RP, we can update the makespan value $T(RP)$ of the current (partial) relaxed plan.

In our ongoing example, suppose that our objective function is $h(S) = f(C(P_S), T(P_S)) = \alpha.C(P_S) + (1 - \alpha).T(P_S)$, $\alpha = 0.55$ and the infinite-lookahead criterion is used to stop the cost propagation process. When we start extracting the relaxed plan, the initial setting is $G = \{at\_la\}$, $RP = \emptyset$ and $RF = \{at\_tucson\}$. Among the three actions $D^{c2}_{t\rightarrow la}$, $T_{lv\rightarrow la}$ and $F_{p\rightarrow la}$ that support the goal at_la, we choose action $A = F_{p\rightarrow la}$ because if we add it to the relaxed plan RP, then the estimated value $h_A(S) = h(RP + A) + h((G \setminus at\_la) \bigcup at\_phx)$ = $(\alpha * C_{exec}(F_{p\rightarrow la}) + (1 - \alpha) * Dur(F_{p\rightarrow la})) + min_t(f(C(at\_phx), t))$ = (0.55*6.0 + 0.45*1.5) + (0.55*1.5 + 0.45*1.5) = 5.475. This is the smallest among the three actions. After we add $F_{p\rightarrow la}$, we update the goal set to $G = \{at\_phx\}$. It is then easy to compare between two actions $D^{c2}_{t\rightarrow phx}$ and $D^{c1}_{t\rightarrow phx}$ to see that $D^{c2}_{t\rightarrow phx}$ is cheaper to achieve *at-phx* given the value $\alpha = 0.55$. The final cost $C(P_S) = 6.0 + 1.5 = 7.5$ and makespan of $T(P_S) = 1.5 + 1.5 = 3$ of the final relaxed plan can be used as the final heuristic estimation $h(S) = 0.55 * 7.5 + 0.45 * 3 = 5.475$ for the given planning problem. [15]

Notice that in the relaxed-plan extraction procedure, we set the time points for the goal set to be the goal deadlines, instead of the latest time points, where the cost functions for the goals stabilized. The reason is that the cost values of facts and actions *monotonically decrease* and the costs are *time-sensitive*. Therefore, the later we set the time points for goals to start searching for the relaxed plan, the better chance we have of getting the low-cost plan, especially when we use the k-lookahead approximation approach with $k \neq \infty$. In our ongoing example, if we use the zero-lookahead algorithm to stop the propagation and found out that the smallest cost is $C(in\_la) = 8.0$ at $t = 2.5$. If we search back for the relaxed plan with the combination $(in\_la, 2.5)$ then we would find a plan $P_1 = (D^{c1}_{t\rightarrow p}, F_{p\rightarrow la})$. However, if we search from the goal deadline, say $t = 7.0$, then we would realize that the lowest cost for the precondition in_phx of $F_{p\rightarrow la}$ at $t = 7.0 - 1.5 = 5.5$ is $C(in\_phx, 5.5) = 1.5$ (caused by $D^{c2}_{t\rightarrow p}$ at time point $t = 2.0$) and thus the final plan is $P_2 = (D^{c2}_{t\rightarrow p}, F_{p\rightarrow la})$ which is cheaper than P_1.

[14] $Precond(A)$ is the set of preconditions of action A.

[15] Notice that because we extract the relaxed plan starting from the goal deadlines, the makespan value is equal to the latest time point at which one of the cost function of a top level goal is updated. That value is decided by the level of lookahead that we use to terminate the propagation. Specifically, zero-lookahead guarantees a smallest makespan for the relaxed plan extracted, and higher levels of lookahead lead to longer makespan values. Nevertheless, regardless of which termination criterion is used, if we want to extract a relaxed plan with the makespan value smaller or equal to a specific value T, then we only need to replace all original goal deadlines by T and use the same extraction process illustrated in Figure 5.



B
package 1
Plane 1
Plane 2
A
package 1
package 2
C
package 2
Relaxed solution with mutex:
move(package1,airplane1,A,B)
move(package2,airplane1,A,C)
Relaxed solution with no mutex:
move(package1,airplane1,A,B)
move(package2,airplane2,A,C)

Figure 6: Example of mutex in the relaxed plan

4.2.1 Improving the relaxed plan heuristic estimation with static mutex relation

When building the relaxed temporal planning graph (RTPG), we ignored the negative interactions between concurrent actions. We now discuss a way of using the static mutex relations to help improve the heuristic estimation when extracting the relaxed plan. Specifically, our approach involves the following steps:

1. Find the set of static mutex relations between the ground actions in the planning problem based on their negative interactions.

2. When extracting the relaxed plan (Section 4.2), besides the orderings between actions that have causal-effect relationship (i.e one action gives the effect that supports the other action's preconditions), we also establish the orderings according to the mutex relations. Specifically, when a new action is added to the relaxed plan, we use the pre-calculated static mutexes to establish ordering between mutually exclusive action pairs so that they can not be executed concurrently. The orderings are selected in such a way that they violate least number of existing causal links in the relaxed plan.

By using the mutex relations, we can improve the makespan estimation of the relaxed plan, and thus the heuristic estimation. Moreover, in many cases, the mutex relations can also help us detect that the relaxed plan is in fact a valid plan, and thus can lead to the early termination of the search. Let's take an example of the Logistics domain that is illustrated in Figure 6. In this example, we need to move two packages from $cityA$ to $cityB$ and $cityC$ and there are two airplanes ($plane_1, plane_2$) at $cityA$ that can be used to move them. Moreover, we assume that $plane_1$ is 1.5 times faster than $plane_2$ and uses the same amount of resources to fly between two cities. There are two relaxed plans

$$P_1 = \{move(package_1, plane_1, cityA, cityB), move(package_2, plane_1, cityA, cityC)\}$$

$$P_2 = \{move(package_1, plane_1, cityA, cityB), move(package_2, plane_2, cityA, cityC)\}$$

that both contain two actions. The first one has shorter makespan if mutexes are ignored. However, if we consider the mutex constraints, then we know that two actions in P_1 can not be executed concurrently and thus the makespan of P_1 is actually longer than P_2. Moreover, the static mutex relations also show that even if we order the two actions in P_1 to be executed sequentially, there is a violation because the first action cuts off the causal link between the initial state and the second one. Thus, the mutex information helps us in this simple case to find a better quality (and consistent) relaxed plan to be used as a heuristic estimation. Here is a sketch of how the relaxed plan P_2 can be found. After the first action $A_1 = move(package_1, plane_1, cityA, cityB)$ is selected to support the goal $at(package_1, cityB)$, the relaxed plan is $RP = A_1$ and the two potential actions to support the second goal $at(package_2, cityC)$ are $A_2 = move(package_2, plane_1, cityA, cityC)$ and $A_2' = move(package_2, plane_2, cityA, cityC)$. With mutex information, we will be able to choose A_2' over A_2 to include in the final relaxed plan.

4.2.2 Using the resource information to adjust the cost estimates

The heuristics discussed in the last two sections have used the knowledge about durations of actions and deadline goals but not resource consumption. By ignoring the resource related effects when building the relaxed plan, we may miss counting actions whose only purpose is to provide sufficient resource-related conditions to other actions.[16] Consequently, ignoring resource constraints may reduce the quality of heuristic estimate based on the relaxed plan. We are thus interested in adjusting the heuristic values discussed in the last two sections to account for the resource constraints.

In real-world problems, most actions consume resources, while there are special actions that increase the levels of resources. Since checking whether the level of a resource is sufficient for allowing the execution of an action is similar to checking the predicate preconditions, one obvious approach to adjust the relaxed plan would be to add actions that provide that resource-related condition to the relaxed plan. However, for many reasons, it turns out to be too difficult to decide which actions should be added to the relaxed plan to satisfy the given resource conditions (for a more elaborate discussion, see [9]). Therefore, we introduce an indirect way of readjusting the cost of the relaxed plan to take into account the resource constraints as follows: We first preprocess the problem specifications and find for each resource *R* an action A_R that can increase the amount of *R* maximally. Let Δ_R be the amount by which A_R increases *R*, and let $C(A_R)$ be the cost value of A_R. Let $Init(R)$ be the level of resource *R* at the state *S* for which we want to compute the relaxed plan, and $Con(R)$, $Pro(R)$ be the total consumption and production of *R* by all actions in the relaxed plan. If $Con(R) > Init(R) + Pro(R)$, we use the following formula to adjust the cost component of the heuristic values according to the resource consumption:

$$C \leftarrow C + \sum_R \left\lceil \frac{(Con(R) - (Init(R) + Pro(R)))}{\Delta_R} \right\rceil * C(A_R)$$

We will call the newly adjusted heuristic *adjusted cost*. The basic idea is that even though we do not know if an individual resource-consuming action in the relaxed plan needs another action to support its resource-related preconditions, we can still adjust the number of actions in the relaxed plan by reasoning about the total resource consumption of *all* the actions in the plan. If we know how much excess amount of a resource *R* the relaxed plan consumes and what is the maximum increment of *R* that is allowed by any individual action in the domain, then we can infer the minimum number of resource-increasing actions that we need to add to the relaxed plan to balance the resource consumption.

[16]In our ongoing example, if we want to drive a car from Tucson to LA and the gas level is low, by totally ignoring the resource related conditions, we will not realize that we need to *refuel* the car before *drive* it.

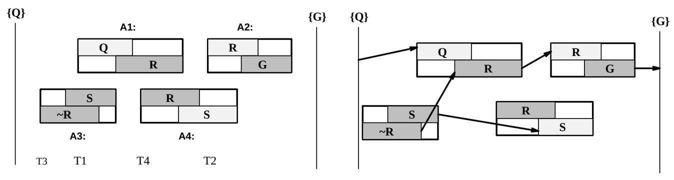


Figure 7: Examples of p.c. and o.c. plans

5 Post-processing to improve Temporal Flexibility

To improve the makespan and execution flexibility of the plans generated by *Sapa*, we post-process and convert them into partially ordered plans. We discuss the details of this process in this section.

Position and Order constrained plans: *A position constrained plan (p.c.) is a plan where the execution time of each action is fixed to a specific time point. An order constrained (o.c.) plan is a plan where only the relative orderings between the actions are specified.*

Sapa produces position constrained plans and the aim of post-processing is to convert them into order constrained ones. Figure 7 shows a valid parallel p.c. plan consisting of four actions A_1, A_2, A_3, A_4 with their starting time points fixed to T_1, T_2, T_3, T_4 and an o.c plan consisting of the same set of actions and achieving the same goals. For each action, the shaded regions show the durations in which each precondition or effect should hold during each action's execution time. The darker ones represent the effect and the lighter ones represent preconditions. For example, action A_1 has a precondition Q and effect R; action A_3 has no precondition and two effects $\neg R$ and S. The arrows show the relative orderings between actions. Those ordering relations represent the o.c plan and thus any execution trace that does not violate those orderings will be a consistent p.c plan.

While generating a p.c. plan consistent with an o.c. plan is easy enough, we are interested in the reverse problem–that of generating an o.c. plan given a p.c. plan. Given a p.c plan P_{pc}, we discuss a strategy that finds a corresponding o.c plan P_{oc} biased to have a reasonably good makespan. Specifically, we extend the explanation-based order generation method outlined in [23] to first compute a causal explanation for the p.c plan and then construct an o.c plan that has just the number of orderings needed to satisfy that explanation. This strategy depends heavily on the positions of all the actions in the original p.c. plan. Thus, it works based on the fact that the alignment of actions in the original p.c. plan guarantees that causation and preservation constraints are satisfied. To facilitate the discussion below, we use the following notations:

- For each (pre)condition p of action A, we use $[st^p_A, et^p_A]$ to represent the duration in which p should hold ($st^p_A = et^p_A$ if p is an instant precondition).
- For each effect e of action A, we use et^e_A to represent the time point at which e occurs.
- For each resource r that is checked for preconditions or used by some action A, we use $[st^r_A, et^r_A]$ to represent the duration over which r is accessed by A.
- The initial and goal states are represented by two new actions A_I and A_G. A_I starts before

all other actions in the P_{pc}, it has no precondition and has effects representing the initial state. A_G starts after all other actions in P_{pc}, has no effect, and has top-level goals as its preconditions.

- The symbol $'' \prec''$ is used through out this section to denote the relative precedence orderings between two time points.

Note that the values of $st^p_A, et^p_A, et^e_A, st^r_A, et^r_A$ are fixed in the p.c plan but are only partially ordered in the o.c plan. The o.c plan P_{oc} is built from a p.c plan P_{pc} as follows:
Supporting Actions: For each precondition p of action A, we choose action A' in the p.c plan to support p such that:

1. $p \in Effect(A')$ and $et^p_{A'} < st^p_A$ in the p.c. plan P_{pc}.
2. There is no action B such that: $\neg p \in Effect(B)$ and $et^p_{A'} < et^{\neg p}_B < st^p_A$ in P_{pc}.
3. There is no other action C that also satisfies the two conditions above and $et^p_C < et^p_{A'}$.

When A' is selected to support p to A, we add the causal link $A' \xrightarrow{p} A$ between two time points $et^p_{A'}$ and st^p_A to the o.c plan. Thus, the ordering $et^p_{A'} \prec st^p_A$ is added to P_{oc}.
Interference relations: For each pair of actions A, A' that interfere with each other, we order them as follows:

1. If $\exists p \in Delete(A') \bigcap Add(A)$, then add the ordering $et^p_A \prec et^{\neg p}_{A'}$ to P_{oc} if $et^p_A < st^{\neg p}_{A'}$ in P_{pc}. Otherwise add $et^{\neg p}_{A'} \prec et^p_A$ to P_{oc}.
2. If $\exists p \in Delete(A') \bigcap Precond(A)$, then add the ordering $et^p_A \prec et^{\neg p}_{A'}$ to P_{oc} if $et^p_A < et^{\neg p}_A$ in P_{pc}. Otherwise, if $et^{\neg p}_{A'} < st^p_A$ in the original plan P_{pc} then we add the ordering $et^{\neg p}_{A'} \prec st^p_A$ to the final plan P_{oc}.

Resource relations: For each resource r that is checked as (pre)condition for action A and used by action A', based on those action's fixed starting times in the original p.c plan P_{pc}, we add the following orderings to the resulted P_{oc} plan as follows:

- If $et^r_A < st^r_{A'}$ in P_{pc}, then the ordering relation $et^r_A \prec st^r_{A'}$ is added to P_{oc}.
- If $et^r_{A'} < st^r_A$ in P_{pc}, then the ordering relation $et^r_{A'} \prec st^r_A$ is added to P_{oc}.

This strategy is backtrack-free due to the fact that the original p.c. plan is correct. All (pre)conditions of all actions in P_{pc} are satisfied and thus for any precondition p of an action A, we can always find an action A' that satisfies the three constraints listed above to support p. Moreover, one of the temporal constraints that lead to the consistent ordering between two interfered actions (logical or resource interference) will always be satisfied because the p.c. plan is consistent and no pair of interfered actions overlap each other in P_{pc}. Thus, the search is backtrack-free and we are guaranteed to find an o.c. plan due to the existence of one legal dispatch of the final o.c. plan P_{oc} (which is the starting p.c. plan P_{pc}). The final o.c. plan is valid because there is a causal-link for every action's precondition, all causal links are safe, no interfering actions can overlap, and all the resource-related (pre)conditions are satisfied. Moreover, this strategy ensures that the orderings on P_{oc} are consistent with the original P_{pc}. Therefore, because the p.c plan P_{pc} is one among multiple p.c plans that are consistent with the o.c plan P_{oc}, the makespan of P_{oc} is guaranteed to be equal or better than the makespan of P_{pc}.

The algorithm discussed in this section is a special case of the partialization problem in metric temporal planning. In [10], we do a systematic study of the general partialization problem and give CSOP (Constraint Satisfaction Optimization Problem) encodings for solving them. The current algorithm can be seen as a particular greedy variable/value ordering strategy for the CSOP encoding.

6 Implementation of *Sapa*

Sapa system with all the techniques described in this paper has been fully implemented in JAVA. The search algorithm for metric temporal planning and other techniques have been incorporated in the current version. Specifically, they include:

1. The forward chaining algorithm (Section 2).
2. The cost sensitive temporal planning graph and the routines to propagate the cost information and extract the heuristic value from it (Section 3).
3. The routines to extract and adjust the relaxed plan with neglected mutex and resource information (Section 4.2).
4. Greedy post-processing routines to extract the causal structure between actions in the plan (Section 5).

The default options for *Sapa* are sum-propagation rule, infinite lookahead termination, resource-adjusted heuristics, and solutions are greedily postprocessed. Beside the techniques described in this paper, we also wrote a JAVA-based parser for PDDL 2.1 Level 3, which is the highest level used in the Third International Planning Competition (IPC3).

To visualize the plans returned by *Sapa* and the relations between actions in the plan (such as causal links, mutual exclusions, and resource relations), we have developed a Graphical User Interface (GUI)[17] for *Sapa*. Figure 8 shows the screen shots of the current GUI. It displays the the time line of the final plan with each action shown with its actual duration and starting time in the final plan. There are options to display the causal relations between actions (found using the greedy approach discussed in Section 5), and logical and resource mutexes between actions. Annotations about the specific times at which individual goals are achieved are also displayed.

Our implementation is publicly available through *Sapa* homepage (*http://rakaposhi.eas.asu.edu/sapa.html*). Given that the planer as well as the GUI are in JAVA, we also provide a web-based interactive access to the planner.

[17]The GUI was developed by Dan Bryce

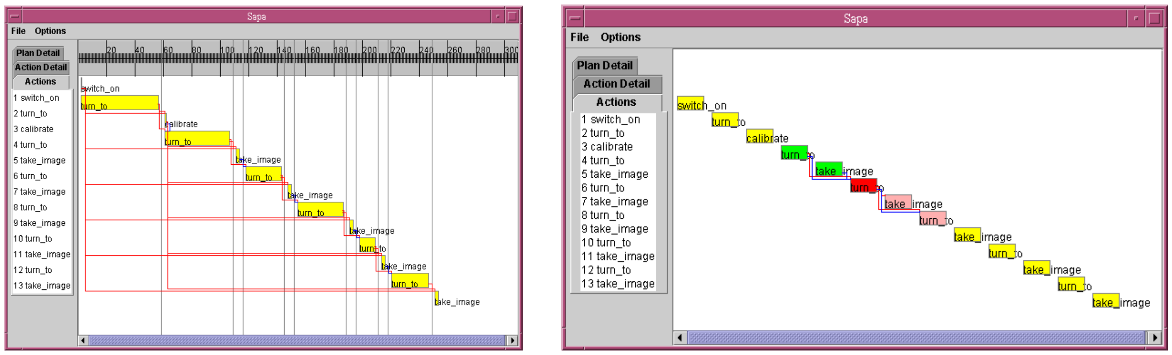


Figure 8: Screen Shots of *Sapa*'s GUI

6.1 Empirical evaluation

We have subjected the individual components of the *Sapa* implementation to systematic empirical evaluation (c.f. [9, 11, 10]). In this section, we will describe the experiments that we conducted to show that *Sapa* [9] is capable of satisfying a variety of cost/makespan tradeoffs. Moreover, we also provide results to show the effectiveness of the heuristic adjustment techniques, the utility of different termination criteria, and the utility of the post-processing. The experiments in this section were run on the Unix SunBlade machine with 256MB of RAM. The evaluation of *Sapa* compared with other systems in the International Planning Competition, with data provided by the competition committee, are provided in the next section.

Our first test suite for the experiments, which show the ability to produce solutions with tradeoffs between time and cost quality, consisted of a set of randomly generated temporal logistics problems provided by Haslum and Geffner [19]. In this set of problems, we need to move packages between locations in different cities. There are multiple ways to move packages, and each option has different time and cost requirements. Airplanes are used to move packages between airports in different cities. Moving by airplanes takes only 3.0 time units, but is expensive, and costs us 15.0 cost units. Moving packages by trucks between locations in different cities costs only 4.0 cost units, but takes longer time of 12.0 time units. We can also move packages between locations inside the same city (e.g. between offices and airports). Driving between locations in one city will cost us 2.0 units and takes 2.0 time units. Load/unload packages into truck or airplane takes 1.0 unit of time and cost 1.0 unit.

We tested with the first 20 problems in the set with the objective function specified as a linear combination of both total execution cost and makespan values of the plan. Specifically, the objective function is set to

$$O = \alpha.C(Plan) + (1 - \alpha).T(Plan)$$

We tested with different values of $\alpha : 0 \leq \alpha \leq 1$. Among the techniques discussed in this paper, we used the sum-propagation rule, infinite look-ahead, and relax-plan extraction using static mutex relation. Figure 9 shows how the average cost and makespan values of the solution change according to the variation of the α value. The results show that the total execution cost of the solution decreases as we increase the α value (thus, giving more weight to the execution cost

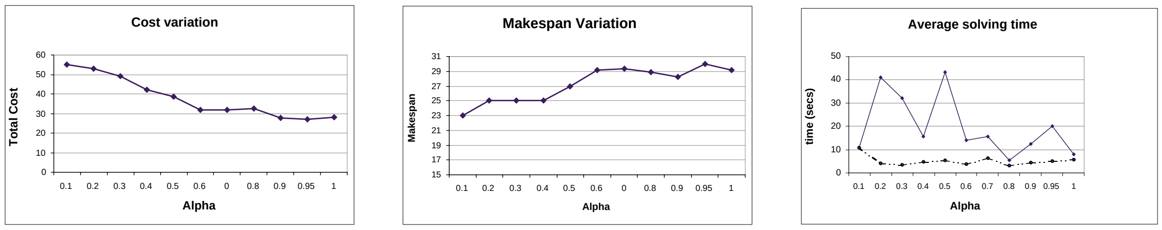


Figure 9: Cost, makespan, and solving time variations according to different weights given to them in the objective function. Each point in the graph corresponds to an average value over 20 problems.

	Relaxed Plan				Relaxed Plan with Adjustment			
Prob	time(s)	node	#act	makespan	time(s)	node	#act	makespan
zeno1	0.272	14/48	5	320	0.317	14/48	5	320
zeno2	92.055	304/1951	23	1020	61.66	188/1303	23	950
zeno3	23.407	200/996	22	890	38.225	250/1221	13	430
zeno4	-	-	-	-	37.656	250/1221	13	430
zeno5	83.122	575/3451	20	640	71.759	494/2506	20	590
zeno6	64.286	659/3787	16	670	27.449	271/1291	15	590
zeno7	1.34	19/95	10	370	1.718	19/95	10	370
zeno8	1.11	27/87	8	320	1.383	27/87	8	320
zeno9	52.82	564/3033	14	560	16.310	151/793	13	590
log-p1	2.215	27/159	16	10.0	2.175	27/157	16	10.0
log-p2	165.350	199/1593	22	18.875	164.613	199/1592	22	18.875
log-p3	-	-	-	-	20.545	30/215	12	11.75
log-p4	13.631	21/144	12	7.125	12.837	21/133	12	7.125
log-p5	-	-	-	-	28.983	37/300	16	14.425
log-p6	-	-	-	-	37.300	47/366	21	18.55
log-p7	-	-	-	-	115.368	62/531	27	24.15
log-p8	-	-	-	-	470.356	76/788	27	19.9
log-p9	-	-	-	-	220.917	91/830	32	26.25

Table 1: Solution times, explored/generated nodes, number of actions, and makespan values of the solutions generated by *Sapa* in the zeno-flying and logistic domains with/without resource adjustment technique.

in the overall objective function). In contrast, when α decreases, thus giving more weight to the makespan, the final cost of the solution increases and the makespan value decreases. The results show that our approach indeed produces solutions that are sensitive to the objective functions that involve both time and cost. The right most graph in the Figure 9 shows the average solution time for this set of experiments. For all the combinations of $\{problem, \alpha\}$, 79% (173/220) are solvable within our limit cutoff time of 300 seconds. The average solution time is 19.99 seconds and 78.61% of the instances can be solved within 10 seconds. The solid line shows the average solution time for different α values. The dashed line shows the average solving time if we take out the 20 (11%) combinations where the solving time is more than 40 seconds (more than two times the average value). We can see that while we generally have larger deviations, solving the multi-objective problems is not significantly costlier than single-objective problems (which corresponds to the end points of the plots).

Utility of resource adjustment technique: In the second set of experiments, we test the utility of the resource adjustment technique discussed in Section 4.2.2 for heuristics discussed in Section 4. We evaluate the performance of *Sapa* on problems from two metric temporal planning domains. The first one is the zeno-flying domain discussed in [32]. The second is our version of the temporal and metric resource version of the logistics domain. In this domain, trucks move packages between locations within one city, and planes carry them from one city to another. Different airplanes and trucks move with different speeds, have different fuel capacities, different fuel-consumption-rates, and different fuel-fill-rates when refueling. The temporal logistics domain is more complicated than the zeno-flying domain because it has more types of resource-consuming actions. Moreover, the *refuel* action in this domain has a dynamic duration, which is not the case for any action in the zeno-flying domain. Specifically, the duration of this action depends on the fuel level of the vehicle and can only be decided at the time we execute that action.

Table 1 shows the running times of *Sapa* for the *relaxed-plan* heuristic with and without metric resource constraint adjustment technique (refer to Section 4.2.2) in the two planning domains discussed above. We tested with 9 problems from each domain. Most of the problems require plans of 10-30 actions, which are quite big compared to problems solved by previous domain-independent temporal planners reported in the literature. The results show that most of the problems are solved within a reasonable time (e.g under 500 seconds). More importantly, the number of nodes (time-stamped states) explored, which is the main criterion used to decide how well a heuristic does in guiding the search, is quite small compared to the size of the problems. In many cases, the number of nodes explored by the best heuristic is only about 2-3 times the size of the plan. Table 1 also shows the number of actions in the solution and the duration (makespan) of the solutions. These categories can be seen as indicative of the problem's difficulty, and the quality of the solutions. By closely examining the solutions returned, we found that the solutions returned by *Sapa* have quite good quality in the sense that they rarely have many irrelevant actions. The absence of irrelevant actions is critical in the metric temporal planners as it will both save resource consumption and reduce execution time. It is interesting to note here that the temporal TLPlan[1], whose search algorithm *Sapa* adapts, usually outputs plans with irrelevant actions. Interestingly, Bacchus & Ady mention that their solutions are still better than the ones

	solving time (secs)			**makespan**			**cost**		
problem	la 0	la 1	la ∞	la 0	la 1	la ∞	la 0	la 1	la ∞
zeno 1	157	0.15	0.16	320	320	320	5	5	5
zeno 2	-	38.40	0.78	-	990	880	-	22	15
zeno 3	1.69	0.31	0.38	650	420	420	17	13	13
zeno 4	-	0.32	0.36	-	420	420	-	13	13
zeno 5	3.25	9.05	2.99	670	690	660	19	20	18
zeno 6	238	0.27	0.33	330	330	330	10	10	10
zeno 7	333	0.27	0.24	450	370	340	10	10	10
zeno 8	177	0.17	0.20	200	200	200	6	6	6
zeno 9	216	0.21	0.24	330	300	300	8	8	8
log 1	424	0.42	0.45	10	10	10	16	16	16
log 2	2.21	171.16	159.12	16.02	19.37	19.37	21	21	21
log 3	-	-	0.99	-	-	10.82	-	-	13
log 4	4.28	0.55	0.62	7.42	7.12	7.12	16	12	12
log 5	-	1.93	2.23	-	12.42	12.42	-	16	16
log 6	2.43	2.28	2.59	16.42	16.42	16.42	21	21	21
log 7	3.64	2.71	34.61	19.32	17.32	26.29	29	27	30
log 8	6.53	3.88	-	17.32	15.32	-	29	27	-
log 9	3.60	3.55	4.01	20.42	20.42	20.42	31	31	31

Table 2: Quality and solving time comparison for different termination criteria.

returned by LPSAT[37], which makes our solutions that much more impressive compared to LPSAT.

Evaluation of different termination criteria: For the same test suite discussed above (i.e. metric temporal zeno-flying and logistics domains), one and infinite lookahead options work similarly and are generally better than zero-lookahead. Table 2 shows the comparison results for zero, one, and infinite lookahead for the set of metric temporal planning problems discussed above. For the objective function, we set $\alpha = 1$, actions costs are 1 unit and action durations are dependent on the distances between locations. In this test suite, the results show that while one and infinite lookahead work similar, they are generally better than zero-lookahead, both in term of solving time and solution quality.

Utility of post-processing: Figure 10 shows the utility of the greedy post-processing technique discussed in Section 5. The test suite contains the same set of problems discussed earlier (i.e. random generated temporal logistics problem provided with TP4[19], and the metric temporal logistics problem). These graphs show the comparisons of makespan values of original parallel position-constrained (p.c) plans and the order-constrained (o.c) plans returned after post-processed. In the left side of Figure 10, we show the comparison between four different makespans for 20 largest problems among the 80 random temporal logistics problems provided

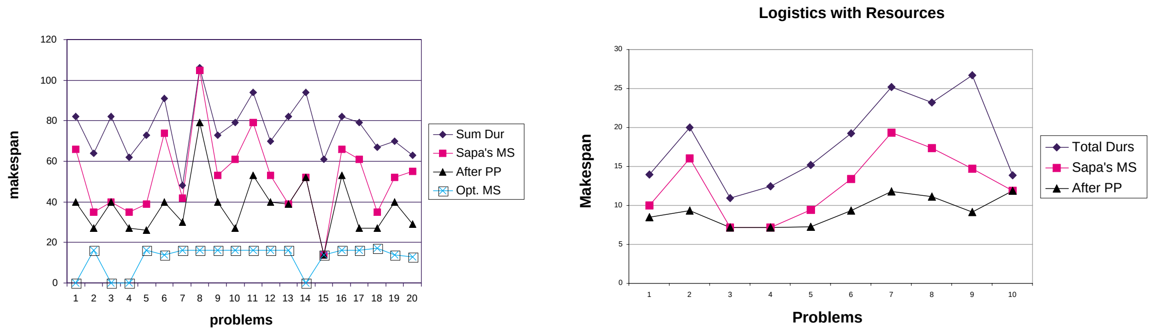


Figure 10: Utility of the greedy post-processing approach for (left) random temporal logistics problem, and (right) metric temporal logistic problems.

with TP4[19] planner. The makespan values are: (1) the optimal makespan (as returned by TGP [35]); (2) the makespan of the plan returned by Sapa; (3) the makespan of the o.c. resulting from the greedy algorithm for partialization discussed in the last section; and (4) the total duration of all actions, which would be the makespan value returned by several serial temporal planners such as AltAlt [29] or GRT [33] if they produce the same solution as Sapa. Notice that the makespan value of *zero* for the optimal makespan indicates that the problem is not solvable by TGP. Compared to the optimal makespan, on the average, the makespan of the serial p.c. plans (i.e, cumulative action duration) is about 4.34 times larger, the makespan of the plans output by Sapa is about 3.23 times larger and the Sapa plans after post processing are about 2.61 times longer (over the set of 75 solvable problems; TGP failed to solve the other 5).

The graph on the right side of Figure 10 shows the utility of the greedy post-processing approach for the set of 10 metric temporal logistics problem discussed earlier in this section. Currently, there is no planner that can handle resources and output optimal makespan. Therefore, we compare only the total duration, the makespan of parallel plans output by Sapa, and the makespan values after partialization. The results show that on average, the backtrack-free value ordering strategy improves the makespan value by 22%.

6.2 *Sapa* in the 2002 International Planning Competition

We entered an implementation of *Sapa*, using several of the techniques discussed in this paper, in the recent International Planning Competition. The specific techniques used in the competition version of *Sapa* are infinite look-ahead termination of cost propagation (Section 3.3), resource adjustment (Section 4.2.2), and greedy post-processing (Section 5). In the competition, we focused solely on the metric/temporal domains. Even though action cost is not part of the standard PDDL 2.1 language used in the competition, infinite-lookahead unit-cost propagation employed in *Sapa* helped it achieve very good performance in problems involving both metric and temporal constraints. The competition results were collected and distributed by the IPC3's organizers and can be found at [13]. Detailed descriptions of domains used in the competition are also available at the same place.

The temporal planning domains in the competition come in two sets, one contains two domains Satellite and Rovers, provided by NASA, representing the applications investigated by them. Figures 11, 12, and 13 show the comparison results in the highest level of PDDL2.1 setting (in terms of the complexities of temporal and metric constraints involved) for the two domains Satellite and Rovers.

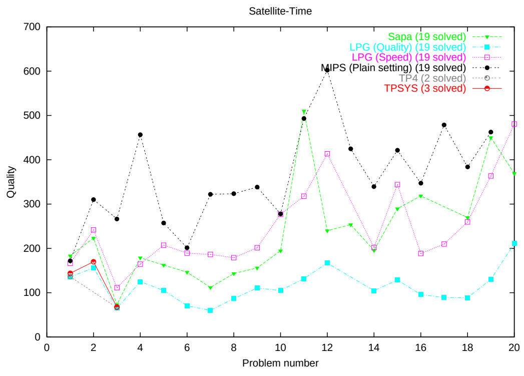
Figure 11 and 12 show that five planners (*Sapa*, LPG, MIPS, TP4, and TPSYS) submitted results for the *timed* setting and only three (*Sapa*, MIPS, TP4) were able to handle the *complex* setting of the Satellite domain. In the *timed* setting, action durations depend on the setting of instruments aboard a particular satellite and the directions it needs to turn to. The *"complex"* setting is further complicated by the fact that each satellite has a different capacity limitation to store only a certain amount of image data. Goals involve taking images of different planets and stars located at different coordinate directions. To achieve the goals, the satellite equipped with the right set of equipment should turn to the right direction, calibrate and take the image.

For the *timed* setting of this Satellite domain, Figure 11 shows that among five planners, *Sapa*, LPG and MIPS were able to solve 19 of 20 problems while TP4 and TPSYS were able to solve 2 and 3 problems respectively. For quality comparison, LPG with *quality* setting was able to return solutions with the best quality, *Sapa* was slightly better than LPG with *speed* setting and was much better than MIPS. LPG with *speed* setting is generally fastest, followed by MIPS and then *Sapa* and LPG with *quality* setting. For the *complex* setting, Figure 12 shows that, among the three planners, *Sapa* was able to solve the most number of problems (16), and generated plans with better quality than MIPS. TP4 produced the best quality solutions, but was able to solve only three smallest problems. The solving times of *Sapa* are slightly higher than MIPS, but are much better than TP4.

The "timed" version of the Rover domain[18] requires a set of scientific analysis to be done using a number of rovers. Each rover carries different set of equipment, and has a different energy capacity. Moreover, each rover can only recharge its battery at certain points (which may be unreachable) that are under the sun. Figure 13 shows that only *Sapa* and MIPS were able to handle the constraints involved with this problem set. *Sapa* again solved more problems (11 vs. 9) than MIPS and also returned better or equal quality solutions in all but one case. The solving time of MIPS is better than *Sapa* in 6 of 9 problems that it returns the solutions.

In the second set of problems which come with temporal constraints, there are three domains: *Depots*, *DriverLogistics* and *Zeno Travel*. *Sapa* participated at the highest level, which is the *"timed"* settings for these three domains. Figure 14 shows the comparison between *Sapa* and three other planners (LPG, MIPS, and TP4) that submitted the results. In this domain, we need to move crates (packages) between different places, the loading actions that place the crates into each truck is complicated by the fact that they need an *empty* hoist. Thus, the *Depot* domain looks like a combination of the original logistics and blockworlds domains. Drive action durations depend on the distances between locations and the speed of the trucks, time for loading the crates depends on the power of the hoist that we use. There is no resource consumption in this highest level. In this domain, *Sapa* was only able to solve 5 problems, compared to 11 by MIPS and 18 by LPG. TP4 solved only one problem. For the problem that *Sapa* was able to solve, the

[18]In the planning competition, each domain had multiple versions–depending on whether or not the actions had durations and whether actions use continuous resource.



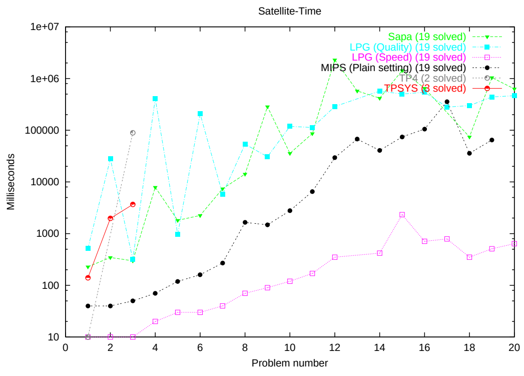
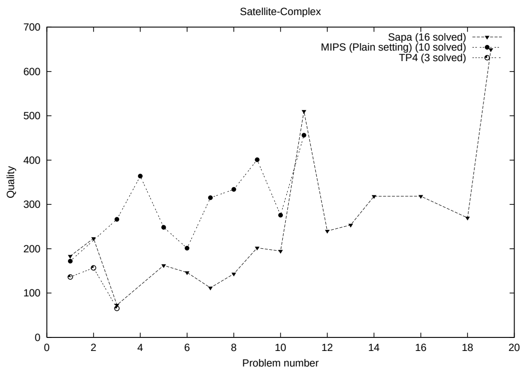


Figure 11: Results for the *time* setting of the Satellite domain (from IPC3 results).



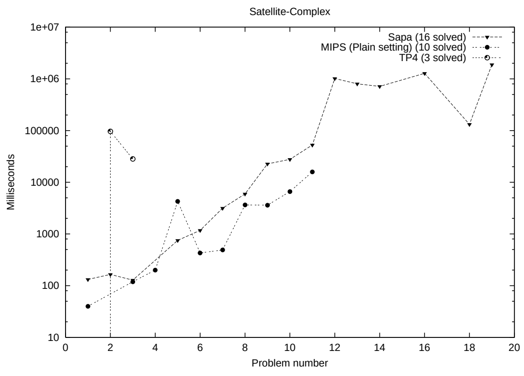
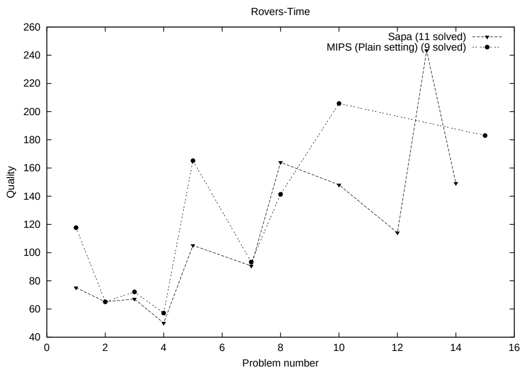


Figure 12: Results for the *complex* setting of the Satellite domain (from IPC3 results).



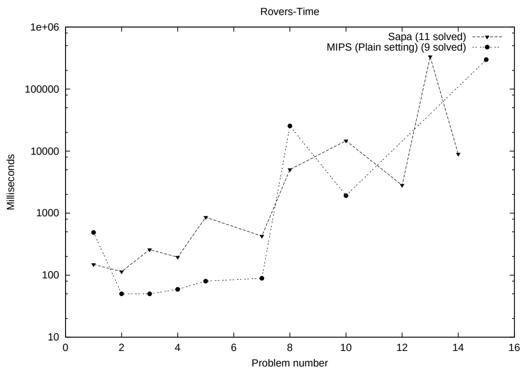


Figure 13: Results for the *time* setting of the Rover domain (from IPC3 results).

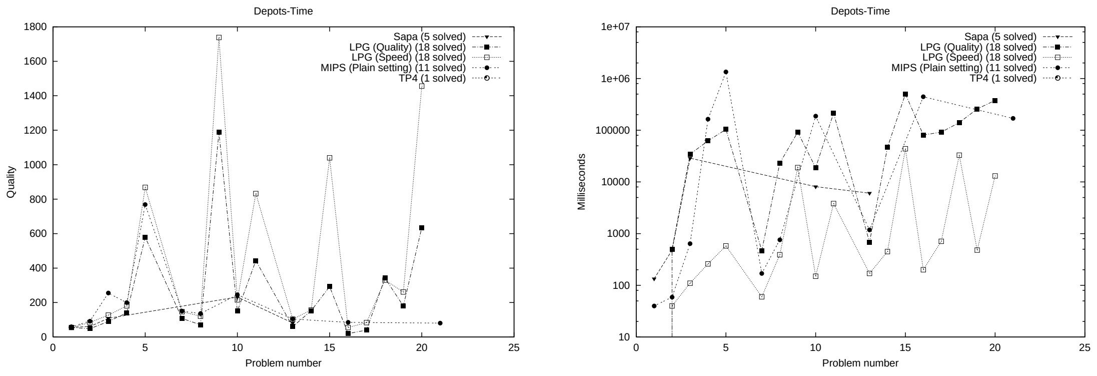


Figure 14: Results for the *time* setting of the Depots domain (from IPC3 results).

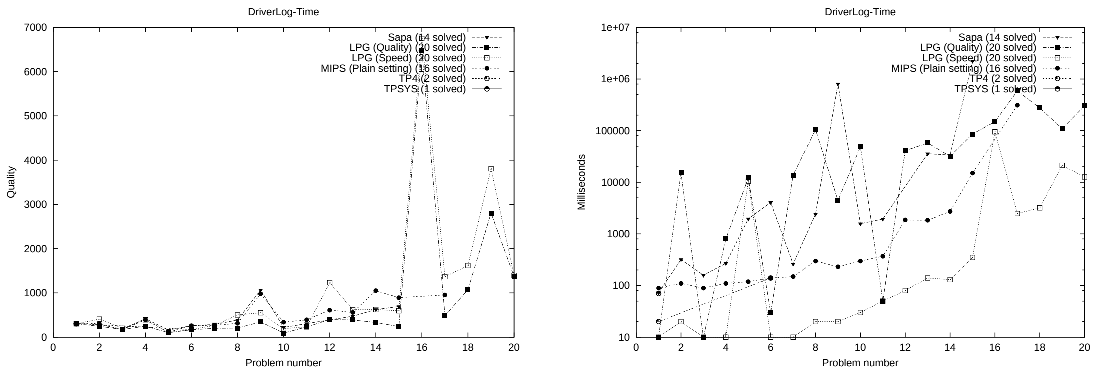


Figure 15: Results for the *time* setting of the DriverLog domain (from IPC3 results).

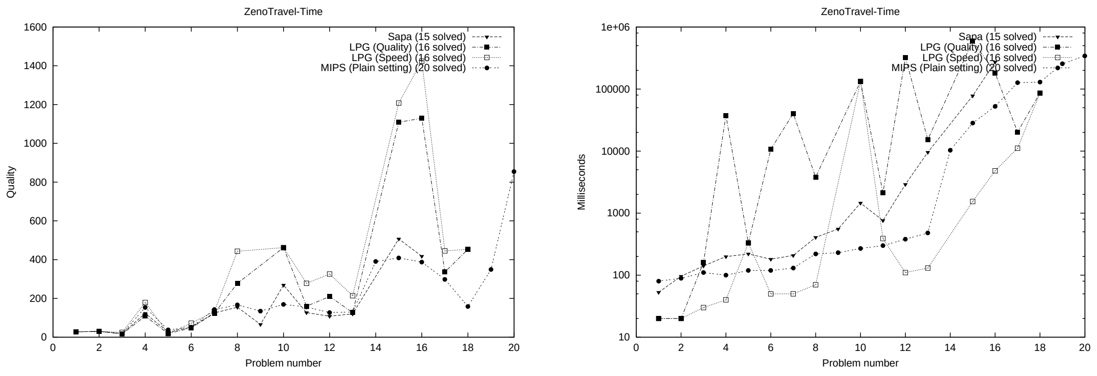


Figure 16: Results for the *time* setting of the ZenoTravel domain (from IPC3 results).

solution quality is as good as other planners. For the speed comparison, LPG with *speed* setting is clearly faster than other planners.

Figure 15 shows how *Sapa* performance compares with other planners in the competition on the *time* setting of the *DriveLog* domain. This is a variation of the original *Logistics* domain in which we do not fly airplanes between different cities but only have to drive trucks between different locations. However, the driver needs to walk between different locations before he can board the truck. Like the *Depot* domain, there is no resource consumption. The durations for walking and driving depend on the specified *time-to-walk* and *time-to-drive* between them. In this domain, *Sapa* solved 14 problems compared to 20 by LPG, 16 by MIPS and 1 by TP4. The quality of the solutions by different planners are very similar. For the speed comparison, LPG with *speed* setting is fastest, then MIPS, then *Sapa* and LPG with *quality* setting.

Finally, Figure 16 shows the performance of *Sapa* in the ZenoTravel domain with *time* setting. In this domain, passengers travel between different cities by airplanes. The airplanes can choose to fly with different speeds (fast/slow), which consuming different amount of fuel for each trip. Airplanes have different fuel capacity and need to refuel if they do not have enough in the tank for each trip. In this domain, *Sapa* and LPG solved 16 problems while MIPS solved 20. The solution quality of *Sapa* and MIPS are similar and in general better than LPG with either *speed* or *quality* setting. LPG with *speed* setting and MIPS solved problems in this domain faster than *Sapa* which in turn faster than LPG with *quality* setting.

In summary, the competition results showed that *Sapa* is one of the best planners in IPC3 in solving problems involving both metric and temporal constraints.

7 Related Work and Discussion

Although there have been several recent domain-independent heuristic planners aimed at temporal domains, most of them have been aimed at makespan optimization, ignoring the cost aspects. For example, both TGP [35] as well as TP4 [19] focus on makespan optimization and ignore the cost aspects of the plan. As we have argued in this paper, ultimately metric temporal planners have to deal with objective functions that are based on both makespan and cost. One recent research effort that recognizes the multi-objective nature of planning is Refanidis' MO-GRT system [34]. On one hand, the MO-GRT approach is more general than our approach in the sense that it deals with the set of non-combinable quality metrics. MO-GRT approach however treats time similar to other consumable resources (with infinite capacity). Temporal constraints on the planning problems (such as when an effect should occur during the course of action), goal deadlines, or the concurrency between actions are *ignored* to scale down the problem to the classical planning assumptions. Like GRT, Metric-FF[22] and MIPS[12] are other forward state space planner that can handle resource constraints. Both of them generate sequential plan. MIPS can then handle durative actions by putting in the action duration and post-process the sequential p.c plans. Multi-Pegg [38] is another recent planner that considers cost-time tradeoffs in plan generation. Multi-Pegg is based on Graphplan approach, and focuses on classical planning problems with non-uniform cost actions. ASPEN [6] is another planner that recognizes the multi-attribute nature of plan quality. ASPEN advocates an iterative repair approach for planning, that assumes the availability of a variety of hand-coded plan repair strategies and their characterization in terms of their effects on the various dimensions of plan quality. Along the lines of ASPEN, LPG[14] is another planner that employs the local search technique over the action-graph. LPG can handle objective functions that involve cost.

Although we evaluated our cost-sensitive heuristics in the context of *Sapa*, a forward chaining planner, the heuristics themselves can also be used in other types of planning algorithms. For example, TGP can be made cost-sensitive by making it propagate the cost functions as part of planning graph expansion. These cost functions can then be used as a basis for variable and value ordering heuristics to guide its backward branch-and-bound search. A similar approach in classical planning has been shown to be successful in [24].

Besides Graphplan-based approaches, our framework can also be used in both forward and backward (state-space, and partial order planners (POP)) to guide the planning search. It is possible due to the fact that directional searches (forward/backward) all need to evaluate the distances between an initial state to the set of temporal goals. In classical planning, works by Nguyen & Kambhampati in [29, 30] have shown that heuristics derived from the classical planning graph can help guiding backward state space and partial order classical planners effectively. Even though the heuristics based on the *level* information in [29, 30] can not directly be extended to use in the temporal planning due to the fact that they assume all goals appear at the same time, our approach is more flexible and does not make such an assumption. The cost functions for facts and actions are based on the continuous time model and are independent of each other. Thus, they can be readily used in the backward planners, especially POP planners, where goals/subgoals are required at different time points.

Our work is also related to other approaches that use planning graphs as the basis for deriving heuristic estimate such as Graphplan-HSP[24], AltAlt[29], RePOP[30], and FF[20]. In the context of these efforts, our contribution can be seen as providing a way to track cost as a function of time on planning graphs. An interesting observation is that cost propagation is in some ways inherently more complex than makespan propagation. For example, once a set of literals enter the planning graph (and are not mutually exclusive), the estimate of the makespan of the shortest plan for achieving them does not change as we continue to expand the planning graph. In contrast, the estimate of the cost of the cheapest plan for achieving them can change until the planning graph levels off. This is why we had to carefully consider the effect of different criteria for stopping the expansion of the planning graph on the accuracy of the cost estimates (Section 3.3).

Another interesting observation is that within classical planning, there was often a confusion between the notions of cost and makespan. For example, the "length of a plan in terms of number of actions" can either be seen as a cost measure (if we assume that actions have unit costs), or a makespan measure (if we assume that the actions have unit durations). These notions get teased apart naturally in metric temporal domains.

In this paper, we concentrated on developing heuristics that can be sensitive to multiple dimensions of plan quality (specifically, makespan and cost). An orthogonal issue in planning with multiple criteria, that we did not explicitly address here, is how the various dimensions of plan quality should be combined during optimization. The particular approach we adopted in our empirical evaluation–viz., considering a linear combination of cost and coverage–is by no means the only reasonable way. Other approaches involve non-linear combinations of the quality criteria, as well as “tiered” objective functions (e.g. rank plans in terms of makespan, breaking ties using cost). A related issue is how to help the user decide the “weights” or “tiers” of the different criteria. Often the users may not be able to articulate their preferences between the various quality dimensions in terms of precise weights. A more standard approach out of this dilemma involves generating all non-dominated plans[19] (the so-called “pareto-set” [7, 31]), and presenting them to the user (unfortunately, often the set of non-dominated plans can be exponential (c.f. [31])). The user is then expected to pick the plan that is most palatable to them. Further, the users may not actually be able to judge the relative desirability of plans when the problems are complex and the plans are long. Thus, a more practicable approach may involve resorting to other indirect methods such as preference elicitation techniques (c.f. [5]).

8 Conclusion

In this paper, we presented *Sapa*, a domain-independent heuristic forward chaining planner that can handle durative actions, metric resource constraints, and deadline goals. *Sapa* is a forward-chaining planner that searches in the space of time-stamped states. It is designed to be capable of handling the multi-objective nature of the metric temporal planning. Our technical contributions included (i) a planning-graph based method for deriving heuristics that are sensitive to both cost and makespan (ii) an easy way of adjusting the heuristic estimates to take the metric resource limitations into account and (iii) a way of post-processing the solution plans to improve their execution flexibility. We described the technical details of extracting the heuristics and presented an empirical evaluation of the current implementation of *Sapa*. An implementation of *Sapa* using a subset of the techniques presented in this paper was one of the best domain independent planners for domains with metric and temporal constraints in the third International Planning Competition, held at AIPS-02.

In the future, we intend to extend *Sapa* in several different directions. First, we want to make it more expressive in term of handling a richer set of temporal and resource constraints, such as exogenous events (e.g a rover can not recharge the battery after sunset). Another direction involves extending our multi-objective search to involve other quality metrics. While we considered cost of a plan in terms of a single monetary cost associated with each action, in more complex domains, the cost may be better defined as a vector comprising the different types of resource consumption. Further, in addition to cost and makespan, we may also be interested in other measures of plan quality such as robustness and execution flexibility of the plan. Our longer term goal is to support plan generation that is sensitive to these extended set of tradeoffs. To this end, we plan to extend our methodology to derive heuristics sensitive to a larger variety of quality measures. Finally, we also plan to consider the issues of planner-scheduler interactions in the context of *Sapa*.

[19]A plan P is said to be dominated by P' if the quality of P' is strictly superior to that of P in at least one dimension, and is better or equal in all other dimensions.

References

[1] Bacchus, F. & Ady, M. 2001. Planning with Resources and Concurrency: A Forward Chaining Approach. In *Proc. of Seventeenth International Joint Conference on Artificial Intelligence*.

[2] Backstrom, C. 1998. Computational Aspects of Reordering Plans. In *Journal of Artificial Intelligence Research 9, 99-137*.

[3] Blum, A. and Furst, M. 1995. Fast planning throught planning graph analysis. In *Proc. of Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*.

[4] Bonet, B., Loerincs, G., and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proc. of Fourteenth National Conference on Artificial Intelligence (AAAI)*

[5] U. Chajewska, L. Getoor, J. Norman and Y. Shahar. 1998. Utility Elicitation as a Classification Problem. In *Proc. of Fourteenth Uncertainty in Artificial Intelligence Conference.*

[6] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D.Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, D.Tran 2000. ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling. In *Proc. of SpaceOps-2000*

[7] P. Dasgupta, P.P. Chakrabarti, and S.C. DeSarkar. Multiobjective Heuristic Search. Vieweg & Son/Morgan Kaufmann, 2001.

[8] Dechter, R., Meiri, I., and Pearl, J. 1990. Temporal Constraint Network. In *Artificial Intelligence Journal 49*.

[9] Do, M. and Kambhampati, S. 2001. Sapa: A domain independent heuristic metric temporal planner. In *Proc. of Sixth European Conference on Planning*

[10] Do, M. and Kambhampati, S. 2002. Improving the Temporal Flexibility of Position Constrained Metric Temporal Planning. In *Temporal Planning Workshop, AIPS-02*

[11] Do, M., and Kambhampati, S. 2002. Planning graph-based heuristics for cost-sensitive temporal planning. In *Proc. of Sixth International Conference on Artificial Intelligence Planning and Scheduling*.

[12] Edelkamp, S. 2001. First Solutions to PDDL+ Planning Problems *In PlanSIG Workshop.*

[13] Fox, M. and Long, D. 2002. Third International Planning Competition. http://www.dur.ac.uk/d.p.long/competition.html

[14] Gerevini, A. and Serina, I. 2002. LPG: A Planner Based on Local Search for Planning Graphs. In *Proc. of Sixth International Conference on Artificial Planning and Scheduling (AIPS-02)*

[15] Long, D. and Fox, M. 1998 Efficient Implementation of the Plan Graph in STAN. In *Journal of AI Research (JAIR), Volume 10, pages 87-115.*

[16] Fox, M. and Long, D. 2001. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Technical Report.*

[17] Garrido, A., Onaindia, E., and Barber, F. 2001. Time-optimal planning in temporal problems. In *Proc. of European Conference on Planning (ECP-01).*

[18] Ghallab, M. and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *Proc. of Second International Conference on Artificial Intelligence Planning and Scheduling*

[19] Haslum, P. and Geffner, H. 2001. Heuristic Planning with Time and Resources In *Proc. of Sixth European Conference on Planning*

[20] Hoffmann, J. & Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. In *Journal of Artificial Intelligence Research* 14:253-302.

[21] Hoffman, J. and Nebel, B. 2001. RIFO Revisited: Detecting Relaxed Irrelevance. In *Proc. of 6th European Conference on Planning.*

[22] Hoffmann, J. 2002. Extending FF to Numerical State Variables. In *Proc. of 15th European Conference on Artificial Intelligence.*

[23] Kambhampati, S. & Kedar, S. 1994. An unified framework for explanation-based generalization of partially ordered and partially instantiated plans. In *Artificial Intelligence Journal 67, 29-70*.

[24] Kambhampati, S. and Nigenda, R. 2000. Distance based goal ordering heuristics for Graphplan. In *Proc. of Fifth International Conference on Artificial Intelligence Planning and Scheduling*.

[25] Koehler, J. 1998. Planning under Resource Constraints. In *Proc. of Eleventh European Conference on Artificial Intelligence*

[26] J. Koehler, B. Nebel, J. Hoffmann, Y. Dimopoulos. Extending Planning Graphs to an ADL Subset In *Proc. of European Conference in Planning.*

[27] Laborie, P. and Ghallab, M. Planning with sharable resource constraints. In *Proc. of Fourteenth International Joint Conference on Artificial Intelligence (IJCAI).*

[28] Muscettola, N. 1994. Integrating planning and scheduling. *Intelligent Scheduling.*

[29] Nguyen, X., Kambhampati, S., and Nigenda, R. 2001. Planning Graph as the Basis for deriving Heuristics for Plan Synthesis by State Space and CSP Search. *In Artificial Intelligence Journal.*

[30] Nguyen, X., and Kambhampati, S., 2001. Reviving Partial Order Plan In *Proc. of Seventeenth International Joint Conference on Artificial Intelligence (IJCAI).*

[31] Papadimitriou C. H. and Yannakakis M. Multiobjective Qury Optimization. ACM Conference on Principles of Database Systems (PODS) 2001.

[32] Penberthy, S. and Well, D. 1994. Temporal Planning with Continuous Change. In *Proc. of 11th National Conference on Artificial Intelligence (AAAI).*

[33] Refanidis, I. and Vlahavas, I. 2001. The GRT Planner: Backward Heuristic Construction in Forward State Space Planning. *Journal of Artificial Intelligence Research, 15.p:115-161.*

[34] Refanidis, I. and Vlahavas, I. 2001. A Framework for Multi-Criteria Plan Evaluation in Heuristic State-Space Planning *Workshop on Planning with Resources, IJCAI-01*.

[35] Smith, D. and Weld, D. 1999. Temporal Planning with Mutual Exclusion Reasoning. In *Proc. of 16th International Joint Conference on Artificial Intelligence (IJCAI)*

[36] Tsamardinos, I., Muscettola, N. and Morris, P. Fast Transformation of Temporal Plans for Efficient Execution. In *Proc. 15th National Conference on Artificial Intelligence (AAAI).*

[37] Wolfman, S. and Weld, D. 1999. The LPSAT system and its Application to Resource Planning. In *Proc. of 16th International Joint Conference on Artificial Intelligence.*

[38] Zimmerman, T. 2002. Generating parallel plans satisfying multiple criteria in anytime fashion. In *Workshop on Planning and Scheduling with Multiple criteria, AIPS-2002*