Improving the Temporal Flexibility of Position Constrained Metric Temporal Plans

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

Techincal Report: TR-02-003

Abstract

In this paper we address the problem of post-processing position constrained plans, output by many of the recent efficient metric temporal planners, to improve their execution flexibility. Specifically, given a position constrained plan, we consider the problem of generating a partially ordered (aka "order constrained") plan that uses the same actions. Although variations of this "partialization" problem have been addressed in classical planning, the metric and temporal considerations bring in significant complications. We develop a general CSP encoding for partializing position-constrained temporal plans, that can be optimized under an objective function dealing with a variety of temporal flexibility criteria, such as makespan. We then present some greedy value ordering strategies that are designed to efficiently generate solutions with good makespan values for these encodings. We demonstrate the effectiveness of our greedy partialization approach in the context of a recent metric temporal planner that produces p.c. plans. We also briefly discuss and evaluate an extension of our partialization approach for temporal plans with resource constraints.

1 Introduction

Of late, there has been significant interest in synthesizing and managing plans for metric temporal domains. Plans for metric temporal domains can be classified broadly into two categories–"position constrained" (p.c.) and "order constrained" (o.c.). The former specify the exactly start time for each of the actions in the plan, while the latter only specify the relative orderings between the actions. The two types of plans offer complementary tradeoffs *vis a vis* search and execution. Specifically, constraining the positions gives complete state information about the partial plan, making it easier to control the search. Not surprisingly, several of the more effective methods for plan synthesis in metric temporal domains search for and generate p.c. plans (c.f. TLPlan[1], Sapa[3], TGP [18]).

At the same time, from an execution point of view, o.c. plans are more advantageous than p.c. plans –they provide better execution flexibility both in terms of makespan and in terms of "scheduling flexibility" (which measures the possible execution traces supported by the plan [20; 15]). They are also more effective in interfacing the planner to other modules such as schedulers (c.f. [19; 11]), and in supporting replanning and plan reuse [21; 9].

A solution to the dilemma presented by these complementary tradeoffs is to search in the space of p.c. plans, but post-process the resulting p.c. plan into an o.c. plan. Although such post-processing approaches have been considered in classical planning ([10; 21; 2]), the problem is considerably more complex in the case of metric temporal planning. The complications include the need to handle the more expressive action representation and the need to handle a variety of objective functions for partialization (in the case of classical planning, we just consider the least number of orderings)

Our contribution in this paper is to first develop a Constraint Satisfaction Optimization Problem (CSOP) encoding for converting a p.c. plan in metric/temporal domains into an o.c. plan. This general framework allows us to specify a variety of objective functions to choose between the potential partializations of the p.c. plan. We then develop a greedy algorithm for partialization, which can be seen as specific variables and value ordering strategies over the CSOP encoding. We will demonstrate the effectiveness of these partialization algorithm in the context of a recent metric/temporal planner called Sapa[3]. Our results show that the temporal flexibility measures, such as the makespan, of the plans produced by Sapa can be significantly improved while retaining Sapa's efficiency advantages.

The paper is organized as follows. In Section 2, we discuss the background on action representation that we assume for the temporal planning problem and the definitions related to the partialization problem. Then, in Section 3 we discuss the CSOP encoding for the partialization problem. Section 4 focuses on how the CSOP encoding can be solved. In Section 4, we also provide a greedy variable and value ordering strategies for solving the encoding. The empirical results for this greedy ordering strategy are provided in Section 5. In Section 6, we show how the partialization encoding can be extended to handle temporal planning problems with continuous changes. Section 7 discusses the related work and Section 8 presents our conclusions.

2 Preliminaries

2.1 Action representation

The representation of actions that we assume in this paper is similar to that used on [1], and [3]. Here, we shall review the temporal aspects of the representation, postponing the discussion of resource consumption aspects to Section 6. Each action A has a duration D_A, starting time st_A, and an end time $et_A = st_A + D_A$. The preconditions of an action may either be instantaneous or durative and their effects may occur at any time point during their execution. Action A has preconditions $p \in Pre(A)$ that may be required to be true for duration $[st^p_A, et^p_A]$ such that $st_A \leq st^p_A \leq et^p_A \leq et_A$. Figure 1 shows graphically the action $A = load(p,t,l)$ ($load(package, truck, location)$). In this action, precondition $p_1 = at(p,l)$ only needs to be true at the starting point of A ($st^{p_1}_A = et^{p_1}_A = st_A$), precondition

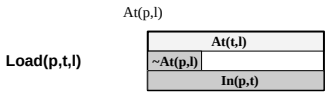

Figure 1: Action Example

$p_2 = at(t,l)$ needs to be true throughout the duration of A ($st_A^{p_2} = st_A, et_A^{p_2} = et_A$)). We need a period of time $d < D_A$ to achieve the effect $e_1 = \neg At(p,l)$, which indicates that the package is not on the ground ($st_A^{e_1} = st_A < et_A^{e_1} < et_A$), and need the whole action duration to achieve the effect $e_2 = In(p,t)$ or having the package inside the truck ($st_A^{e_2} = st_A < et_A^{e_2} = et_A$). For each effect e of action A that occurs at et_A^e, there is a duration $[st_A^e, et_A^e] : st_A \leq st_A^e \leq et_A^e \leq et_A$ in which we do not allow any process that leads to the event that causes $\neg e$ to occur. Note that unlike preconditions, an effect e can both be positive (add) or negative (delete).

An important issue in converting a p.c. plan into an o.c. plan is to ensure that actions that are not ordered with respect to each other are free of any interference. In general, two actions A and A' interfere if $\exists p : \neg p \in E(A) \wedge (p \in P(A') \bigcup E(A')$. Unlike classical planning, the temporal concurrency between A and A' depends on the exact temporal constraints (values of st^p, et^p in A and A'). Thus, the temporal relations between two interfering actions A and A' depend on the exact proposition p that relates them and it is possible to have more than one interference relation between two actions, each one enforcing different set of temporal constraints. Therefore, we use the notation $\bigotimes_{A,A'}^{p}$ to denote a specific interference relation between A, A' as it holds if $\neg p \in E(A)$ and $p \in P(A') \bigcup E(A')$. Each interference relation $\bigotimes_{A,A'}^{p}$ constrains the temporal orders between A and A' - specifically with the constraint $(et_A^{\neg p} < st_{A'}^{p}) \vee (et_{A'}^{p} < st_A^{\neg p})$. In our example in Figure 1, if any action A' that uses the proposition $At(p,l)$ of having the effect of causing $At(p,l)$, then A' is interference upon p with action $A = load(p,t,l)$.

2.2 Problem Definition

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.*

There are two types of position constrained plans: *serial* and *parallel*. In a serial position constrained plan, no concurrency is allowed. In a parallel position constrained plan, actions are allowed to execute concurently. Examples of the serial p.c. plans are the ones returned by classical planners such as GRT [17], MIPS [5] and their temporal cousins. The parallel p.c. plans are the ones returned by Graphplan-based planners and their temporal cousins such as Sapa[3], TGP[18], TP4[7]. Examples of planners that output order constrained (o.c.) plans are Zeno[16], HSTS[14], IxTexT[11].

Figure 2 shows a valid p.c. parallel 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 lighters represent preconditions. For example, action A_1 has a precondition Q and effect R; action A_3 has no preconditions and two effects $\neg R$ and S.

It should be easy to see that o.c. plans provide more execution flexibility than p.c. plans. In particular, an o.c. plan can be "dispatched" for execution in any way consistent with the relative ordering among the actions. In other words, for each valid o.c. plan P_{oc}, there may be multiple valid p.c. plans that satisfy the orderings in P_{oc}, which can be seen as different ways of dispatching the o.c. plan.

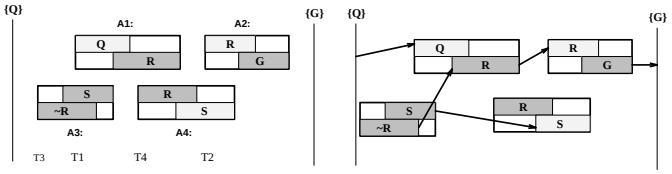

Figure 2: Examples of p.c. and o.c. plans

A measure of the temporal quality of a plan is its "makespan." The **makespan** of a plan is the minimum time needed to execute a plan. For a p.c. plan, the makespan is the duration from the earliest starting time until the latest ending time among all actions. In the case of serial p.c. plans, it is easy to see that the makespan will be greater than or equal to the sum of the durations of all the actions in the plan. For the o.c. plan, the makespan is the minimum makespan of any of the p.c. plans that are consistent with it. Given an o.c. plan P_{oc}, there is a polynomial time algorithm based on topological sort of the orderings in P_{oc}, which outputs a p.c. plan P_{pc} where all the actions are assigned earliest possible start time point according to the ordering in P_{oc}. The makespan of that p.c. plan P_{pc} is then used as the makespan of the original o.c. plan P_{oc}.

While generating a p.c. plan consistent with an o.c. plan is easy enough, in this paper, we are interested in the reverse problem–that of generating an o.c. plan given a p.c. plan. Thus, for a given p.c. plan P_{pc}, we want to find the optimal o.c. plan according to some criterion of temporal/execution flexibility such as smallest makespan or smallest number of orderings. In the next section, we shall provide a general CSP encoding for this "partialization problem." Finding optimal solution for this encoding turns out to be NP-hard even for classical planning (i.e., non-durative actions)[2]. Consequently, we shall develop value ordering strategies that are able to find a reasonable solution for the encoding in polynomial time.

3 Formulating a CSOP encoding for the partialization problem

Suppose that P_{pc}, containing a set of actions $\mathcal{A}$, and their starting times st_A^{pc}, is a valid p.c. plan for some temporal planning problem $\mathcal{P}$. Let P_{oc} be a partialization of P_{pc} for the problem $\mathcal{P}$. P_{oc} must then satisfy the following conditions:

1. P_{oc} contains the same actions $\mathcal{A}$ as P_{pc}.
2. P_{oc} is executable. This requires that the preconditions of all actions are satisfied, and no pair of interfering actions are allowed to execute concurrently.
3. P_{oc} is a valid plan for $\mathcal{P}$. This requires that P_{oc} satisfies all the top level goals (including deadline goals) of $\mathcal{P}$.
4. (Optional) The orderings on P_{oc} are such that P_{pc} is a legal dispatch (execution) of P_{oc}.
5. (Optional) The set of orderings in P_{oc} is minimal (i.e., no ordering is redundant)

Given that P_{oc} is an order constrained plan, ensuring goal and precondition satisfaction involves ensuring that (a) there is a causal support for the condition and that (b) the condition, once supported, is not violated by any possibly intervening action. The fourth constraint ensures that P_{oc} is in some sense an *order generalization* of P_{pc} [10]. This is not strictly needed if our

interest is only to improve temporal flexibility.[1] Finally, the fifth **c**onstraint above is optional in the sense that any objective function defined in terms of the orderings anyway ensures that P_{oc} contains no redundant orderings.

In the following, we will develop a CSP encoding for finding P_{oc} that captures the constraints above. This involves specifying the variables, their domains, and the inter-variable constraints.

Variables: The encoding will consist of both continuous (temporal) and discrete variables. The continuous variables represent the temporal aspects of actions in the plan, and the discrete variables represent the logical causal structure and orderings between the actions in the plan. Specifically, for the set of actions in the p.c. plan P_{pc} and two additional actions A_i and A_g representing the initial and final dummy actions,[2] the set of variables are as follows:

Temporal variables: For each action A, the encoding has one variable st_A to represent the time point at which we can start executing A. The domain for this variable is $Dom(st_A) = [0, +\infty)$.

Discrete variables: There are several different types of discrete variables representing the causal structure and qualitative orderings between actions:

- *Causal effect:* We need variables to specify the causal links relationship between actions. Specifically, for each fact $p \in P(A)$ and a set of actions $\{B_1, B_2.....B_n\}$ such that $p \in E(B_i)$, we set up one variable: S^p_A where $Dom(S^p_A) = \{B_1, B_2....B_n\}$.
- *Supportively related:* Two actions A and A' are supportively related if $\exists p \in (E(A) \bigcap P(A')$. For each such pair, we introduce one variable $\bigcirc^p_{AA'}$: $Dom(\bigcirc^p_{AA'}) = \{\prec, \perp\}$ ($A\ before_p\ A$', or *no-order* between A & A'). In our example in Figure 2, some ordering variables are: $\bigcirc^R_{A_1A_2}$, $\bigcirc^R_{A_3A_2}$, $\bigcirc^S_{A_3A_4}$.
- *Destructively related (interference):*Two actions A and A' are destructively related if they interfere with each other. For each such pair, using the same notation introduced at the end of Section 2.1, we introduce one variable $\bigotimes^p_{AA'}$: $Dom(\bigotimes^p_{AA'}) = \{\prec, \succ\}$ ($A\ before_p\ A'$, or $A\ after_p\ A'$). For the plan in Figure 2, the ordering variables are: $\bigotimes^R_{A_1A3}$ and $\bigotimes^R_{A_2A_3}$.[3]

Following are the necessary constraints to represent the relations between different variables:

1. If B supports the condition p for A, then there should be a supportive ordering between B and A w.r.t. p:
   $S^p_A = B \Rightarrow \bigcirc^p_{BA} = \prec$
2. Causal link protections: If B supports p to A, then every other action A' that has an effect $\neg p$ must be prevented from coming between B and A:
   $S^p_A = B \Rightarrow \forall A',\ \neg p \in E(A') : (\bigotimes^p_{A'B} = \prec) \vee (\bigotimes^p_{A'A} = \succ)$
3. Constraints to prevent non-minimal orderings (optional): If B does not support p to A, then there is no need for a supportive ordering between B and A w.r.t. p:
   $S^p_A \neq B \Rightarrow \bigcirc^p_{BA} = \perp$.
4. There are constraints between ordering variables and action start time variables (as per the discussion in Section 2.1). Specifically, we want to enforce that if $A \prec_p A'$ then $et^p_A < st^p_{A'}$. However, because we only maintain one continuous variable st_A in the encoding for each action, the constraints are as follow:
   $\bigcirc^p_{AA'} = \prec \Leftrightarrow\ st_A + (et^p_A - st_A) < st_{A'} + (st^p_{A'} - st_{A'})$.
   $\bigotimes^p_{AA'} = \prec \Leftrightarrow\ st_A + (et^{\neg p}_A - st_A) < st_{A'} + (et^p_{A'} - st_{A'})$.
   $\bigotimes^p_{AA'} = \succ \Leftrightarrow\ st_{A'} + (et^p_{A'} - st_{A'}) < st_A + (st^{\neg p}_A - st_A)$.
   Notice that all values $(st^p_A - st_A)$, $(et^p_A - st_A)$ are constants for all actions A and propositions p.
5. Deadlines and other temporal constraints: These model any deadline type constraints in terms of the temporal variables. For example, if all the goals need to be achieved before time t_g, then we need to add a constraint: $st_{A_g} \leq t_g$. Other temporal constraints, such as those that specify that certain actions should be executed before/after certain time points, can also be handled by adding similar temporal constraints to the encoding.
6. Constraints to make the orderings on P_{oc} consistent with P_{pc} (optional): Let T_A be the fixed starting time point of action A in the original p.c plan P_{pc}. To guarantee that P_{pc} is consistent with the set of orderings in the resulting o.c plan P_{oc}, we add a constraint to ensure that the value T_A is always present in the live domain of the temporal variable st_A.

Given the presence of both discrete and temporal variables in this encoding, the best way to handle it is to view it as a leveled CSP encoding where in the satisficing assignments to the discrete variables activate a set of temporal constraints between the temporal variables. These temporal constraints, along with the deadline and order consistency constraints are represented as a temporal constraint network [4]. Solving the network involves making the domains and inter-variable intervals consistent across all temporal constraints [20]. The consistent temporal network then represents the o.c. plan. Actions in the plan can be executed in any way consistent with the temporal network (thus providing execution flexibility).

Objective Function: Each satisficing assignment for the encoding above will correspond to a possible partialization of P_{pc}, i.e., an o.c. plan that contains all the actions of P_{pc}. However, some of these assignments (o.c. plans) may have better execution properties than the others. We can handle this by specifying an objective function to be optimized, and treating the encoding as a Constraint Satisfaction Optimization (CSOP) encoding. The only requirement on the objective function is that it be specifiable in terms of the variables of the encodings. Objective functions such as makespan minimization and order minimization readily satisfy this requirement.

4 Solving the partialization encoding

As mentioned above, the encoding, once setup, can be solved by a coupled framework (such as the one used in LPSAT [22]) where in a discrete CSP solver is used to handle the discrete variables, and a temporal CSP solver is used to handle the temporal variables. Every assignment to the discrete variables will activate a set of constraints between the temporal variables, which, in conjunction with the constraints of type 4 and 5 can

[1] In the terminology of [2], the presence of fourth constraint ensures that P_{oc} is a de-ordering of P_{pc}, while in its absence P_{oc} can either be a de-ordering or a re-ordering.

[2] A_i has no preconditions and has effects that add the facts in the initial state. A_g has no effect and has preconditions representing the goals.

[3] Sometimes, we will use the notation $A \prec_p A'$ to represent $\bigcirc^p_{AA'} = \prec$ and $\bigotimes^p_{AA'} = \prec$.

be solved by the temporal CSP solver. All the temporal constraints are "simple" [4] and can thus be handled in terms of a simple temporal network. Optimization can be done using a branch and bound scheme on top of this.

Notwithstanding the foregoing discussion, solving the CSOP encoding will be NP-hard problem (this follows from [2]). Consequently, we focus on developing variable and value ordering strategies for the encoding, which can ensure that the very first satisficing solution found will have a high quality in terms of the objective function. Clearly, these strategies will depend on the specific objective function. In the following, we will develop strategies that are suited to objective functions based on minimizing the makespan.

4.1 Greedy value ordering strategies for solving the encoding

In this section, we discuss a value ordering strategy that finds an assignment to the CSOP encoding such that the corresponding o.c plan P_{oc} is biased to have a reasonably good makespan. The 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 causality and preserving constraints are satisfied. Specifically, all CSP variables are assigned values as follows:

Supporting Variables: For each variable S^p_A representing the action that is used to support precondition p of action A, we choose action A' such that:

1. $p \in E(A')$ and $et^p_{A'} < st^p_A$ in the p.c. plan P_{pc}.
2. There is no action B s.t: $\neg p \in E(B)$ and $et^p_{A'} < et^{\neg p}_B < st^p_A$ in P_{pc}.
3. There is no other action C that also satisfies two conditions above and $et^p_C < et^p_{A'}$.

Interference ordering variables: For each variable $\bigotimes^p_{AA'}$, we assign value:

1. $\bigotimes^p_{AA'} = \prec$ if $et^p_A < st^p_{A'}$ in P_{pc}.
2. $\bigotimes^p_{AA'} = \succ$ if $et^p_{A'} < st^p_A$ in P_{pc}.

Other ordering variables: For all the ordering variables $\bigcirc^p_{AA'}$ that are not enforced to have value $\prec$ by the assignments to supporting variables $S^p_{AA'}$, we assign values $\bigcirc^p_{AA'} = \perp$.

This strategy is backtrack-free due to the fact that the original p.c. plan is correct. Thus, all preconditions of all actions are satisfied and for all *supporting variables* we can always find an action A' that satisfies the three constraints listed above to support a precondition p of action A. More over, one of the temporal constraints that lead to the assignment of *interference ordering variables* $\bigotimes^p_{AA'}$ will always be satisfied because the p.c. plan is consistent and no pair of actions that have interference relations overlap each other. Finally, this strategy ensures that the orderings on P_{oc} are consistent with the original P_{pc}. Therefore, the search is backtrack-free and no constraint is violated because there is one legal dispatch of the final o.c. plan P_{oc}, which is the starting p.c. plan P_{pc}. Moreover, 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 P_{pc}.

Complexity: It is also easy to see that the complexity of the greedy algorithm is $O(S * A + I + O)$ where S is the number of supporting relations, A is the number of actions in the plan, I is the number of interference relations and O is the number of ordering variables. In turn $S \leq A * P$, $I \leq A^2$ and $O \leq P * A^2$ where P is the number of preconditions of an action. Thus, the complexity of the algorithm is $O(P * A^2)$.

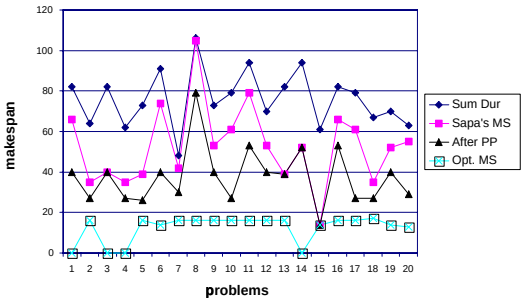

Figure 3: Temporal Logistics with *drive inter-city* actions.

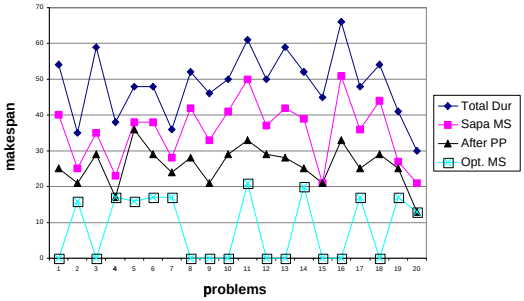

Figure 4: Temporal Logistics without *drive inter-city* actions.

5 Empirical results for Temporal Planning

We have implemented the variable and value ordering discussed in the last section (Section 4.1) and tested it with the Sapa planner. Sapa is a forward state space planner that outputs parallel p.c. plans. The results reported in [3] show that while Sapa is quite efficient, it often generates plans with inferior makespan values. Our aim is to see how much of an improvement our partialization algorithm provides for the plans produced by Sapa. The test suite is the 80 random temporal logistics provided with TP4 planner. In this planning domain trucks move packages between locations inside a city and airplanes move them between cities. Figure 3 and 4 show the comparison results for only the 20 largest problems, in terms of number of cities and packages, among 80 of that suite. In Figure 3, trucks are allowed to move packages between different locations in different cities, while in the Figure 4, trucks are not allowed to do so.

The graphs show the comparison between four different makespan values: (1) the optimal makespan (as returned by TGP [18]); (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 GRT [17], or MIPS [5] 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.

For the first test which allows driving between cities action, 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). For the second test, without the driving inter-city actions. The comparison results with regard to optimal solutions are: 2.39 times longer for serial plans, 1.75 times longer for the plans output by

Sapa, and 1.31 times longer after partialization. These results are averaged over the set of 69 out of the 80 problems that were solvable by TGP.[4]

Thus, the partialization algorithm improves the makespan values of the plans output by Sapa by an average of 20% in the first set and 25% in the second set. Notice also that the same technique can be used by GRT [17] or MIPS [5] and in this case, the improvement would be 40% and 45% respectively for the two problem sets.

The partialization and topological sort times are extremely short. Specifically, they are less than 0.1 seconds for all problems with the number of actions ranging from 16 to 37. Thus, using our partialization algorithm as a post-processing stage essentially preserves the significant efficiency advantages of planners such as Sapa, GRT and MIPS, that search in the space of p.c. plans, while improving the temporal flexibility of the plans generated by those planners.

Finally, it should be noted that partialization improves not only makespan but also other temporal flexibility measures. For example, the "scheduling flexibility" of a plan defined in [15], which measures the number of actions that do not have any ordering relations among them, is significantly higher for the partialized plans, compared even to the parallel p.c. plans generated by TGP. In fact, our partialization routine can be applied to the plans produced by TGP to improve their scheduling flexibility.

6 Temporal planning with continuous changes

An advantage of setting up an encoding for the partialization problem is that the encoding can be generalized to handle other types of constraints on the plan. In fact, we have extended the encoding to handle temporal problems in the presence of metric/resource consumption. In this section, we briefly summarize the extension, and present some preliminary empirical results.

Extending the Representation: Let V_t^r be a value of a metric resource r at a time point t, we assume that an action A that uses r has following constraints:

1. **Resource duration:** Like propositions, A uses r for a period between two time points st_A^r and et_A^r s.t: $st_A \leq st_A^r \leq et_A^r \leq et_A$.
2. **Resource preconditions:** At time point st_A^r, there may be some constraints on the value of r (such as the action should have enough resources to be executetable). We assume that the constraints are in the form of comparison such as: $V_t^r \diamond K$, where K is a constant, and $\diamond \in \{<, >, \leq, \geq, =\}$. To simplify the discussion, for the rest of this section, we will only discuss the case for $\diamond = >$. The other cases are very similar.
3. **Resource effects:** Actions may increase/decrease an amount U_A^r of r during the period from st_A^r to et_A^r.

Extending the encoding: To be able to output the o.c plan that is resource-consistent, which means that all resource related constraints $V_{st_A^r}^r > K$ are satisfied by the set of orderings in the o.c plan, we need to introduce a new set of variables and constraints to our general CSOP encoding discussed in previous sections. The details are as follows:

Variables: For each pair of actions A and A' that use the same resource r, we introduce one variable $\bigodot_{AA'}^r$ to represent the resource-enforced ordering between them (similar to the way $\bigcirc_{AA'}^r$ represents the precondition enforced orderings; see Section 3). If A and A' can not use the same resource at the same time, then $Dom(\bigodot_{AA'}^r) = \{\prec, \succ\}$, otherwise $Dom(\bigodot_{AA'}^r) = \{\prec, \succ, \perp\}$.

Constraints: There are two additional types of constraints:

1. Constraints representing the relations between the resource-related orderings and action start time variables:
   $\bigodot_{AA'}^r = \prec \Leftrightarrow st_A + (et_A^r - st_A) < st_{A'} + (st_{A'}^r - st_{A'})$
   $\bigodot_{AA'}^r = \succ \Leftrightarrow st_{A'} + (et_{A'}^r - st_{A'}) < st_A + (st_A^r - st_A)$
   Notice that the values $(st_A^r - st_A)$ and $(et_A^r - st_A)$ are constants for a given action A that uses resource r.
2. Constraints to guarantee the resource consistency for all actions: Specifically, for a given action A that uses resource r and has a resource constraint $V_{st_A^r}^r > K$, let $\{A_1, A_2, ...A_n\}$ be a set of actions that also use r and $Init_r$ be the value of r at the initial state. We set up one constraint that involves all variables $\bigodot_{A_i A}^r$ as follows:

$$Init_r + \sum_{A_i \prec_r A} U_{A_i}^r + \sum_{A_i \perp_r A, U_{A_i}^r < 0} U_{A_i}^r > K$$

   (where $A_i \prec_r A$ is shorthand for $\bigodot_{A_i A}^r = \prec$). The constraint above ensures that regardless of how the actions A_i that have no ordering relation with A ($\bigodot_{A_i A}^r = \perp$) are aligned temporally with A, the orderings between A and other actions guarantee that A has enough resource ($V_{st_A^r}^r > K$) to execute.

Greedy value ordering: Beside the default variable and value ordering used by any solver that we choose to solve our CSOP encoding, we can also use the value ordering similar to the strategy used to assign values to the causal and ordering variables in Section 4.1. Specifically, the variables $\bigodot_{AA'}^r$ can be assigned values based on their fixed starting times in the original p.c plan P_{pc} as follows:

- $\bigodot_{AA'}^r = \prec$ if $et_A^r < st_{A'}^r$ in P_{pc}.
- $\bigodot_{AA'}^r = \succ$ if $et_{A'}^r < st_A^r$ in P_{pc}.
- $\bigodot_{AA'}^r = \perp$ otherwise.

Due to the fact that the original p.c plan P_{pc} is correct, it is easy to see that the value ordering discussed above will lead to a backtrack-free search over the set of resource-related ordering variables (do not cause any temporal or resource inconsistency).

Preliminary Empirical Evaluation: We implemented this value ordering strategy and tested it with a set of logistics problems in which different trucks and airplanes consume fuel at different rates while moving packages. They also need to refuel when they do not have enough fuel in their tank to finish

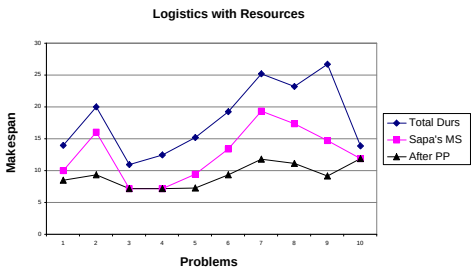

Figure 5: Results for temporal problems with resources

[4] While TGP could not solve several problems in this test suite, Sapa is able to solve all 80 of them.

the trip. We tested with 10 problems and the results are shown in Figure 5. 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%.

7 Related Work

The complementary tradeoffs provided by the p.c. and o.c. plans have been recognized in classical planning. One of the earliest efforts to attempt to improve the temporal flexibility of plans was the work by Fade and Regnier [6] who discussed an approach for removing redundant orderings from the plans generated by STRIPS system. Later work by Mooney [13] and Kambhampati and Kedar [10]characterized this partialization process as one of explanation-based order generalization. Backstrom [2] categorized approaches for partialization into "de-ordering" approaches and "re-ordering" approaches. The order generalization algorithms fall under the de-ordering category. He was also the first to point out the NP-hardness of maximal partialization, and to characterize the previous algorithms as greedy approaches.

The work presented in this paper can be seen as a principled generalization of the partialization approaches to metric temporal planning. Our novel contributions include: (1) providing a CSP encoding for the partialization problem and (2) characterizing the greedy algorithms for partialization as specific value ordering strategies on this encoding. In terms of the former, our partialization encoding is general in that it encompasses both de-ordering and re-ordering partializations–based on whether or not we include the optional constraints to make the orderings on P_{oc} consistent with P_{pc}. In terms of the latter, the work in [21] and [10] can be seen as providing a greedy value ordering strategy over the partialization encoding for classical plans. However, unlike the strategies we presented in Sections 4.1 and 6, their value ordering strategies are not sensitive to any specific optimization metric.

It is interesting to note that our encoding for partialization is closely related to the so-called "causal encodings" [8]. Unlike casual encodings, which need to consider supporting a precondition or goal with every possible action in the action library, the partialization encodings only need to consider the actions that are present in P_{pc}. In this sense, they are similar to the encodings for replanning and plan reuse described in [12]. Also, unlike causal encodings, the encodings for partialization demand optimizing rather than satisficing solutions. Finally, in contrast to our encodings for partialization which specifically handle metric temporal plans, causal encodings in [8] are limited to classical domains.

8 Conclusion

In this paper we addressed the problem of post-processing position constrained metric temporal plans to improve their execution flexibility. We developed a general CSP encoding for partializing position-constrained temporal plans, that can be optimized under an objective function dealing with a variety of temporal flexibility criteria, such as makespan. We then presented greedy value ordering strategies that are designed to efficiently generate solutions with good makespan values for these encodings. We evaluated the effectiveness of our greedy partialization approach in the context of a recent metric temporal planner that produces p.c. plans. Our results demonstrate that the partialization approach is able to provide between 25-40% improvement in the makespan, with extremely little overhead. We also briefly discussed an extension of our partialization approach for temporal plans with resource constraints, and demonstrated empirically that partialization can lead to up to 22% improvement of the makespan. Currently, we are focusing on developing greedy value ordering strategies that are sensitive to other types of temporal flexibility measures besides makespan.

References

[1] Bacchus, F. and Ady, M. 2001. Planning with Resources and Concurrency: A Forward Chaining Approach. *Proc IJCAI-2001*.

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

[3] Do, M., and Kambhampati, S. 2001. Sapa: A Domain-Independent Heuristic Metric Temporal Planner. *Proc ECP-01*

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

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

[6] Fade, B. and Regnier, P. 1990 Temporal Optimization of Linear Plans of Action: A Strategy Based on a Complete Method for the Determination of Parallelism *Technical Report*

[7] Haslum, P. and Geffner, H. 2001. Heuristic Planning with Time and Resources. *Proc ECP-2001*

[8] Kautz, H., McAllester, D. and Selman B. Encoding Plans in Propositional Logic *In Proc. KR-96.*

[9] Ihrig, L., Kambhampati, S. Design and Implementation of a Replay Framework based on a Partial order Planner. *Proc. AAAI-96.*

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

[11] Laborie, P. and Ghallab, M. Planning with sharable resource constraints. *Proc IJCAI-95*.

[12] Mali, A. Plan Merging and Plan Reuse as Satisfiability *Proc ECP-99*.

[13] Mooney, R. J. Generalizing the Order of Operators in Macro-Operators *Proc. ICML-1988*

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

[15] Nguyen, X., and Kambhampati, S. 2001. Reviving Partial Order Planning. *Proc IJCAI-01*.

[16] Penberthy, S. and Weld, D. 1994. Planning with Continous Changes. *Proc. AAAI-94*

[17] Refanidis, I. and Vlahavas, I. 2001. Multiobjective Heuristic State-Space Planning *Technical Report.*

[18] Smith, D. & Weld, D. Temporal Planning with Mutual Exclusion Reasoning. *Proc IJCAI-99*

[19] Srivastava, B., Kambhampati, S., and Do, M. 2001. Planning the Project Management Way: Efficient Planning by Effective Integration of Causal and Resource Reasoning in RealPlan. *Artificial Intelligence Journal 131*.

[20] Tsamardinos, I., Muscettola, N. and Morris, P. Fast Transformation of Temporal Plans for Efficient Execution. *Proc. AAAI-98.*

[21] Veloso, M., Perez, M, & Carbonell, J. 1990. Nonlinear planning with parallel resource allocation. *Workshop on Innovative Approaches to Planning, Scheduling and Control*.

[22] Wolfman, S. and Weld, D. 1999. The LPSAT system and its Application to Resource Planning. *In Proc. IJCAI-99.*