From: ICAPS-03 Proceedings. Copyright © 2003, AAAI (www.aaai.org). All rights reserved.

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

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 propose several approaches (e.g. coupled CSP, MILP) of solving this encoding. We also present a greedy value ordering strategy that is 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 compare the effects of greedy and optimal partialization using MILP encodings on the set of metric temporal problems used at the Third International Planning Competition.

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 exact 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(Bacchus & Ady 2001), Sapa(Do & Kambhampati 2001), TGP (Smith & Weld 1999), MIPS(Edelkamp 2001)). 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 (Tsamardinos et. al. 1998; Nguyen & Kambhampati 2001)). They are also more effective in interfacing the planner to other modules such as schedulers (c.f. (Srivastava et. al. 2001; Laborie & Ghallab 1995)), and in supporting replanning and plan reuse (Veloso et. al. 1990; Ihrig & Kambhampati 1996).

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 ((Kambhampati & Kedar 1994; Veloso et. al. 1990; Backstrom 1998)), 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. Among several approaches to solve this CSOP encoding, we will discuss in detail the one approach that converts it to an equivalent MILP encoding, which can then be solved using any MILP solver such as CPLEX or LPSolve to produce an o.c. plan optimized for some objective function. Our intent in setting up this encoding was not to solve it to optimum–since that is provably NP-hard (Backstrom 1998)–but to use it for baseline characterization of greedy partialization algorithms. The greedy algorithms that we present can themselves be seen as specific variable and value ordering strategies over the CSOP encoding. We will demonstrate the effectiveness of these greedy partialization algorithms in the context of our metric/temporal planner called *Sapa*(Do & Kambhampati 2001; 2002). 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 effi-

*We thank David E. Smith and the ICAPS reviewers for useful comments on the earlier drafts of this paper. This research is supported in part by the NSF grant IRI-9801676, and the NASA grants NAG2-1461 and NCC-1225.
Copyright © 2003, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

ciency advantages. The greedy partialization algorithms developed in this paper were used as part of the *Sapa* implementation that took part in the 2002 International Planning Competition (Fox & Long 2002). At the competition, *Sapa* was one of the best planners for metric temporal domains, both in terms of time and in terms of quality. The partialization procedures clearly helped the quality of the plans produced by *Sapa*. We also show that at least for the competition domains, the option of solving the encodings to optimum, is not particularly effective in improving the makespan further.

The paper is organized as follows. First, we provide the definitions related to the partialization problem. Then, we discuss the CSOP encoding for the partialization problem and focus on how the CSOP encoding can be solved. In We also provide a greedy variable and value ordering strategies for solving the encoding. Later, we provide the empirical results for this greedy ordering strategy and the optimal partialization using MILP encodings. Finally, we discuss the related work and present our conclusions.

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 concurrently. Examples of the serial p.c. plans are the ones returned by classical planners such as AltAlt(Nguyen et. al. 2001), HSP(Bonet & Geffner 1997), FF(Hoffmann 2000), GRT (Refanidis & Vlahavas 2001). The parallel p.c. plans are the ones returned by Graphplan-based planners and the temporal planners such as *Sapa* (Do & Kambhampati 2001), TGP(Smith & Weld 1999), TP4(Haslum & Geffner 2001). Examples of planners that output order constrained (o.c.) plans are Zeno(Penberthy & Weld 1994), HSTS(Muscettola 1994), IxTeT(Laborie & Ghallab 1995).

Figure 1 shows, on the left, 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 on the right, an o.c plan consisting of the same set of actions and achieving the same goals. For each action, the marked rectangular regions show the durations in which each precondition or effect should hold during each action's execution time. The shaded rectangles represent the effects and the white ones represent preconditions. For example, action A_1 has a precondition Q and effect R and 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 orderings 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.

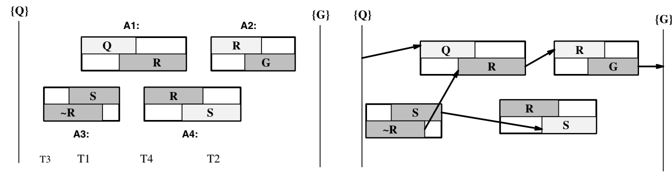


Figure 1: Examples of p.c. and o.c. plans

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.

Partialization: *Partialization is the process of generating a valid order constrained plan P_{oc} from a set of actions in a given position constrained plan P_{pc}.*

We can use different criteria to measure the quality of the o.c. plan resulting from the partialization process (e.g. makespan, slack, number of orderings). One important criterion is a plan's "makespan." The *makespan* of a plan is the minimum time needed to execute that plan. For a p.c. plan, the makespan is the duration between the earliest starting time and 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 an 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 orderings 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}.

Formulating a CSOP encoding for the partialization problem

In this section, we develop a general CSOP encoding for the partialization problem. The encoding contains both continuous and discrete variables. The constraints in the encoding guarantee that the final o.c plan is consistent, executable, and achieves all the goals. Moreover, by imposing different user's objective functions, we can get the optimal o.c plan by solving the encoding.

Preliminaries

Let P_{pc}, containing a set of actions $\mathcal{A}$ and their fixed starting times st^{pc}_A, be a valid p.c. plan for some temporal planning problem $\mathcal{P}$. We assume that each action A in P_{pc} is in the standard PDDL2.1 Level 3 representation (Fox & Long 2001).[1] To facilitate the discussion on the CSOP encoding in the following sections, we will briefly discuss the action representation and the notation used in this paper:

- 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 instantaneous precondition).
- For each effect e of action A, we use et^e_A to represent the time point at which e occurs.

[1]PDDL2.1 Level 3 is the highest level used in the Third International Planning Competition.

- 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 preconditions and has effects representing the initial state. A_G starts after all other actions in P_{pc}, has no effects, 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 CSOP encoding for the partialization problem

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 (pre)conditions 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., all ordering constraints are non-redundant, in that they cannot be removed without making the plan incorrect).

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} (Kambhampati & Kedar 1994). In the terminology of (Backstrom 1998), 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. This is not strictly needed if our interest is only to improve temporal flexibility. Finally, the fifth constraint 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 and discrete variables. The continuous variables represent the temporal and resource aspects of the actions in the plan, and the discrete variables represent the logical causal structure and orderings between the actions. Specifically, for the set of actions in the p.c. plan P_{pc} and two additional dummy actions A_i and A_g representing the initial and goal states,[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 $Dom(st_A) = [0, +\infty)$.

Resource variables: For each action A and the resource $r \in R(A)$, we use a pseudo variable[3] V^r_A to represent the value of r (resource level) at the time point st^r_A.

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 link relationships between actions. Specifically, for each condition $p \in P(A)$ 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 $Dom(S^p_A) = \{B_1, B_2....B_n\}$.
- *Interference:* Two actions A and A' are in logical interference on account of p if $p \in Precond(A) \cup Effect(A)$ and $\neg p \in Effect(A')$. For each such pair, we introduce one variable $I^p_{AA'}$: $Dom(I^p_{AA'}) = \{\prec, \succ\}$ ($A\ before_p\ A'$, or $A\ after_p\ A'$). For the plan in Figure 1, the interference variables are: $I^R_{A_1A_3}$ and $I^R_{A_2A_3}$. Sometimes, we will use the notation $A \prec_p A'$ to represent $I^p_{AA'} = \prec$.
- *Resource ordering:* For each pair of actions A and A' that use the same resource r, we introduce one variable $R^r_{AA'}$ to represent the resource-enforced ordering between them. If A and A' can not use the same resource concurrently, then $Dom(R^r_{AA'}) = \{\prec, \succ\}$, otherwise $Dom(R^r_{AA'}) = \{\prec, \succ, \perp\}$. Sometimes, we will use the notation $A \prec_r A'$ to represent $R^p_{AA'} = \prec$.

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

1. 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:
$S^p_A = B \Rightarrow \forall A',\ \neg p \in E(A') : (I^p_{A'B} = \prec) \vee (I^p_{A'A} = \succ)$
2. Constraints between ordering variables and action start time variables: 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 need to be posed as follows:
$$I^p_{AA'} = \prec \Leftrightarrow st_A + (et^{\neg p}_A - st_A) < st_{A'} + (st^p_{A'} - st_{A'}).$$
$$I^p_{AA'} = \succ \Leftrightarrow st_{A'} + (et^p_{A'} - st_{A'}) < st_A + (st^{\neg p}_A - st_A).$$
$$R^p_{AA'} = \prec \Leftrightarrow st_A + (et^r_A - st_A) < st_{A'} + (st^r_{A'} - st_{A'}).$$
$$R^r_{AA'} = \succ \Leftrightarrow st_{A'} + (et^r_{A'} - st_{A'}) < st_A + (st^r_A - st_A).$$
Notice that all values $(st^{p/r}_A - st_A)$, $(et^{p/r}_A - st_A)$ are constants for all actions A, propositions p, and resource r.
3. Constraints to guarantee the resource consistency for all actions: Specifically, for a given action A that has a resource constraint $V^r_{st^r_A} > K$, let U^r_A be an amount of resource r that A produces/consumes ($U^r_A > 0$ if A produces r and $U^r_A < 0$ if A consumes r). Suppose that

[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] We call V a pseudo variable because the constraints involving V are represented not directly, but rather indirectly by the constraints involving U^r_A; see below.

$\{A_1, A_2, ...A_n\}$ is the set of actions that also use r and $Init_r$ be the value of r at the initial state, we set up a constraint that involves all variables $R^r_{A_iA}$ as follows:

$$Init_r + \sum_{A_i \prec_r A} U^r_{A_i} + \sum_{A_i \perp_r A, U^r_{A_i} < 0} U^r_{A_i} > K \quad (3)$$

(where $A_i \prec_r A$ is a shorthanded notation for $R^r_{A_iA} = \prec$). The constraint above ensures that regardless of how the actions A_i that have no ordering relations with A ($R^r_{A_iA} = \perp$) are aligned temporally with A, the orderings between A and other actions guarantee that A has enough resource ($V^r_{st^r_A} > K$) to execute.

Note that in the constraint (3) above, the values of $U^r_{A_i}$ can be static or dynamic (i.e. depending on the relative orderings between actions in P_{oc}). Let's take the actions in the IPC3's *ZenoTravel* domain for example. The amount of fuel consumed by the action $fly(cityA, cityB)$ only depends on the fixed distance between $cityA$ and $cityB$ and thus is static for a given problem. However, the amount of fuel $U^{fuel}_{refuel} = capacity(plane) - fuel(plane)$ produced by the action $A = refuel(plane)$ depends on the fuel level just before executing A. The fuel level in turn depends on the partial order between A and other actions in the plan that also consume/produce $fuel(plane)$. In general, let $U^r_A = f(f_1, f_2, ...f_n)$ (3.1) where f_i are functions that have values modified by some actions $\{A^i_1, A^i_2 ... A^i_m\}$ in the plan. Because all A^i_k are mutex with A according to the PDDL2.1 specification, there is a resource ordering variable $R^{f_i}_{AA_i}$ with $Dom(R^{f_i}_{AA_i}) = \{\prec, \succ\}$ and the value $V^{f_i}_{st_A}$ can be computed as:

$$V^{f_i}_{st_A} = Init_r + \sum_{A_i \prec_{f_i} A} U^{f_i}_{A_i} \quad (3.2)$$

Then, we can subtitute the value of $V^{f_i}_{st_A}$ in equation (3.2) for each variable f_i in (3.1). Solving the set of equations (3.1) for all action A and resource r, we will find the value of U^r_A. Finally, that value of U^r_A can be used to justify the consistency of the CSP constraint (3) for each resource-related precondition $V^r_{st^r_A} > K$. Other constraints $V^r_{st^r_A} \star K$ ($\star = \leq, \geq, <$) are handled similarly.

4. 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 (e.g $L \leq st_A \leq U$).
5. 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.

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 is specifiable in terms of the variables of the encodings. Objective functions such as makespan minimization and order minimization readily satisfy this requirement. Following are several objective functions that worth investigating:

Temporal Quality:

- Minimum Makespan: $minimize\ Max_A(st_A + dur_A)$
- Maximize summation of slacks:

$$Maximize \sum_{g \in Goals} (st^g_{A_g} - et^g_A) \ : \ S^g_{A_g} = A$$

- Maximize average flexibility: $Maximize\ Average(Domain(st_A))$

Ordering Quality:

- Fewest orderings: $minimize\ \#(st_A \prec st_{A'})$

Solving the partialization encoding

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 (Dechter et. al. 1990). Solving the network involves making the domains and inter-variable intervals consistent across all temporal constraints (Tsamardinos et. al. 1998). 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). All the temporal constraints are "simple" (Dechter et. al. 1990) 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.

Although the leveled CSP framework is a natural way of solving this encoding, unfortunately, there are no off-the-shelf solvers which can support its solution. Because of this, for the present, we convert the encoding into a Mixed Integer Linear Programming (MILP) problem, so it can be solved using existing MILP solvers, such as LPSolve and CPLEX. In the following, we discuss the details of the conversion into MILP.

We remind the readers that as mentioned in the introduction, the purpose of setting up the MILP conversion was not to use it as a practical means of partializing the p.c. plans, but rather to use it as a baseline for evaluating the greedy algorithms–which will be presented in the later section.

Optimal Post-Processing Using MILP Encoding

Given the CSOP encoding discussed in the previous section, we can convert it into a Mixed Integer Linear Program (MILP) encoding and use any standard solver to find an optimal solution. The final solution can then be interpreted to get back the o.c plan. In this section, we will first discuss the set of MILP variables and constraints needed for the encoding, then, we concentrate on the problem of how to setup the objective functions using this approach.

MILP Variables and Constraints For the corresponding CSOP problem, the set of variables and constraints for the MILP encoding is as follows:

Variables: We will use the the binary integer variables (0,1) to represent the logical orderings between actions and linear variables to represent the starting times of actions in the CSOP encoding.

- *Binary (0,1) Variables:*
  1. Causal effect variables: $X^p_{AB} = 1$ if $S^p_A = B$, $X^p_{AB} = 0$ otherwise.
  2. Mutual exclusion (mutex) variables: $Y^p_{AB} = 1$ if $I^p_{AB} = \prec$, $Y^p_{BA} = 1$ if $I^p_{AB} = \succ$,
  3. Resource interference variables: $X^r_{AA'} = 1$ if $A \prec_r A'$ (i.e. $et^r_A < st^r_{A'}$). $N^r_{AA'} = 1$ if there is no order between two actions A and A' (they can access resource r at the same time).[4]
- *Continuous Variable:* one variable st_A for each action A and one variable st_{A_g} for each goal g.

Constraints: The CSP constraints discussed in the previous section can be directly converted to the MILP constraints as follows:

- Mutual exclusion: $Y^p_{AB} + Y^p_{BA} = 1$
- Only one supporter: $\forall p \in Precond(A) : \Sigma X^p_{BA} = 1$
- Causal-link protection: $\forall A', \neg p \in Effect(A') : (1 - X^p_{AB}) + (Y^p_{A'A} + Y^p_{BA'}) \geq 1$
- Ordering and temporal variables relation: $M.(1 - X^p_{AB}) + (st^p_B - et^p_A) > 0$; where M is a very big constant.[5]
- Mutex and temporal variables relation: $M.(1 - Y^p_{AB}) + (st^p_B - et^p_A) > 0$
- Resource-related constraints: Let U^r_A be the amount of resource r that the action A uses. $U^r_A < 0$ if A consumes (reduces) r and $U^r_A > 0$ if A produces (increases) r. For now, we assume that U^r_A are constants for all actions A in the original p.c plan returned by *Sapa* and will elaborate on this matter in the later part of this section.
  - Only one legal ordering between two actions: $X^r_{AA'} + X^r_{A'A} + N^r_{AA'} = 1$
  - Resource ordering and temporal ordering relations: $M.(1 - X^r_{AA'}) + (st^r_{A'} - et^r_A) > 0$
  - Constraints for satisficing resource-related preconditions:
  $$Init_r + \sum X^r_{A'A}.U^r_{A_i} + \sum_{U^r_B < 0} N^r_{AB}.U^r_B > K \quad (4)$$
  if the condition to execute action A is that the resource level of r when A starts executing is higher than K.[6]
- Constraints to enforce that all actions start after A_{init} and finish before A_{goal}: $\forall A : st_A - st_{A_{init}} \geq 0, st_{A_{goal}} - (st_A + dur_A) \geq 0$.
- Goal deadline constraints: $st_{A_g} \leq Deadline(g)$

Note that in the equation (4) listed above, we assume that U^r_A are all constants for all resource-related functions r and actions A. The reason is that if U^r_A are also variables (non-constant), then equation (4) is no longer a linear equation (and thus can not be handled by a MILP solver). In the previous section, however, we discussed the cases in which the values of U^r_A are not constants and depend on the relative orders between A and other actions in the plan. Therefore, to use the MILP approach, we need to add additional constraints to ensure that the values of U^r_A are all constants and equal to the U^r_A values in the original p.c plan. By doing so, we in some sense enforce that the actions in P_{oc} and P_{pc} are physically identical in terms of the reasources they produce/consume.

To ensure that U^r_A are constants and consistent with the orderings in the final o.c plan P_{oc}, we have to do some pre-processing and add additional linear constraints to the MILP encoding. First, we pre-process P_{pc} and for each action A and function f which A accesses/changes the value of, we record the value $V^f_{st^f_A}$ and U^f_A. Let's call those fixed values $VPC^f_{st^f_A}$ and UPC^f_A. Then, for each action A and function f which A accesses the value, we add the following MILP constraint to the encoding:

$$V^r_{st^r_A} = Init_r + \sum X^f_{A_iA}.UPC^f_{A_i} = VPC^f_{st^f_A} \quad (4.2)$$

The linear constraint (4.2) means that the orderings between A and other actions that change the value of f ensure that the value of f when we execute A is $V^f_{st^f_A} = VPC^f_{st^f_A}$. Then, using equation (3.1), the value of U^r_A can be calculated as:

$$U^r_A = f(f_1, ...f_n) = f(VPC^{f_1}_{st^{f_1}_A}, ...VPC^{f_1}_{st^{f_n}_A}) = UPC^r_A$$

and is fixed for every pair of action A and resource r regardless of the orderings in the final o.c plan P_{oc}.

[4] In PDDL 2.1, two actions A and B are allowed to access the same function (resource) overlappingly if: (1) A do not change any function that B is checking as its precondition; (2) A and B using the functions to change the value of r in a *commute* way (increase/decrease only).

[5] The big constant M enforces the logical constraint: $X^p_{AB} = 1 \Rightarrow et^p_A < st^p_B$. Notice that if $X^p_{AB} = 0$ then no particular relation is needed between et^p_A and st^p_B. In this case, the objective function would take care of the actual value of et^p_A and st^p_B. The big M value can be any value which is bigger than the summation of the durations of all actions in the plan.

[6] This constraint basically means that even if the actions that has no ordering with A ($N^r_{AA'} = 1$) align with A in the worst possible way, the A has enough r at its starting time. Notice also that the initial level of r can be considered as the production of the initial state action A_{init}, which is constrained to execute before all other actions in the plan.

MILP Objective Functions Starting from the base encoding above, we can model a variety of objective functions to get the optimal o.c. plans upon solving MILP encoding as follows:
Minimum Makespan:

- An additional (continuous) variable to represent the plan makespan value: V_{ms}
- Additional constraints for all actions in the plan: $\forall A: \ st_A + dur_A \leq V_{ms}$
- MILP Objective function: $minimize \ V_{ms}$

Maximize minimum slack[7] *value:*

- An additional (continuous) variable to represent the minimum slack value: V_{ms}
- Additional constraints for all goals: $\forall g \forall A: \ V_{ms} - (M.X^g_{AA_g} + (st_{A_g} - et^g_A)) \geq 0$, M is a very big constant. This constraint contains two parts. The first part: $M.X^g_{AA_g} + (st_{A_g} - et^g_A)$ guarantees that among all actions that add g (cause g for A_g), the real supporting action A ($X^g_{AA_g} = 1$) is the one that is used to measure the slack value (i.e. among all actions that can potentially support goal g, the value $M.X^g_{AA_g} + (st_{A_g} - et^g_A)$ is biggest for A choosen to support g). The whole equation with V_{ms} involved would then guarantee that the slack value is measured correctly. The same big M value is used across all the constraints for different goals and would be subtracted from the final V_{ms} value to get the correct *minimum slack* value.
- MILP objective function: $minimize \ V_{ms}$

Minimum number of orderings:

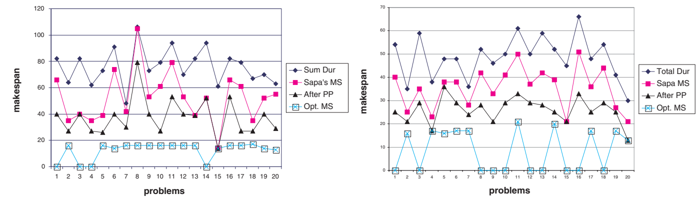
- Additional binary ordering variables for every pair of actions: O_{AB}
- Additional constraints: $\forall A, B, p: O_{AB} - X^p_{BA} \geq 0, O_{AB} - Y^p_{AB} \geq 0$
- MILP objective function: $minimize \Sigma O_{AB}$

Greedy value ordering strategies for solving the encoding

Solving the CSOP encoding to optimum, whether by MILP encoding or otherwise, will be NP-hard problem (this follows from (Backstrom 1998)). Our motivation in developing the encoding was not to solve it to optimum, but rather to develop greedy 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. The optimal solutions can be used to characterize how good the solution found by greedy variable/value ordering procedure.

Clearly, the best greedy variable/value ordering strategies will depend on the specific objective function. In this section, we will develop strategies that are suited to objective functions based on minimizing the makespan. Specifically, we discuss a value ordering strategy that finds an assignment

[7] The objective function of *maximize maximum slack* and *maximize summation of slack* can be handled similarly.



(a) With *drive inter-city* action. (b) Without *drive inter-city* action.

Figure 2: Compare different makespan values for random generated temporal logistics problems

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 preservation 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 the two conditions above and $et^p_C < et^p_{A'}$.

Interference ordering variables: For each variable $I^p_{AA'}$, we assign values based on the fixed starting times of A and A' in the original p.c plan P_{pc} as follows:

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

Resource variables: For each variables $R^r_{AA'}$, we assign values based on the fixed starting times of A and A' in the original p.c plan P_{pc} as follows:

- $R^r_{AA'} = \prec$ if $et^r_A < st^r_{A'}$ in P_{pc}.
- $R^r_{AA'} = \succ$ if $et^r_{A'} < st^r_A$ in P_{pc}.
- $R^r_{AA'} = \perp$ otherwise.

This strategy is backtrack-free due to the fact that the original p.c. plan is correct. Thus, all (pre)conditions 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. Moreover, one of the temporal constraints that lead to the consistent ordering between two interfering actions (logical or resource interference) will always be satisfied because the p.c. plan is consistent and no pair of interfering 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}.

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 $O(P * A^2)$.

Empirical Evaluation

We have implemented the greedy variable and value ordering discussed in the last section and have also implemented the MILP encoding discussed previously. We tested our implementation with the *Sapa* planner. *Sapa* is a forward state space planner that outputs parallel p.c. plans. The results reported in (Do & Kambhampati 2001) 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.

Given a p.c plan P_{pc}, the greedy partialization (GP) and optimal partialization (OP) routines return three different plans. The first is what we call a *logical order constrained (logical o.c)* plan. It consists of a set of logical relations between actions (e.g. causal link from the end point of A_1 to the start point of A_2). The logical relations include (i) causal link, (ii) logical mutex, and (iii) resource mutex. The second is a *temporal order constrained (temporal o.c)* plan in which the temporal o.c plan is represented by the temporal relations between the starting time points of actions. This in effect collapses multiple logical relations (in a logical o.c plan) between a pair of actions (A_1, A_2) into a single temporal relation between A_1 and A_2. The temporal o.c plan is actually a Simple Temporal Network (STN) (Dechter et. al. 1990).[8] The third plan is the p.c plan that is a legal dispatch of the logical or temporal o.c plan, in which each action is given an earliest starting time allowed by the logical/temporal ordering in P_{oc}. The makespan of this p.c plan is the minimal makespan of any dispatch of P_{oc} and is thus reported as the makespan after post-processing.

In the next three sections, we report the empirical results for the greedy partialization approach and the optimal partialization using MILP approach. The MILP solver that we used is the Java version of the lp_solve package[9]. Since this solver is also implemented in Java, integrating it into the *Sapa* package was somewhat easier.

[8]While logical o.c plan gives more information, the temporal o.c plan is simpler and more compact. Moreover, from the flexibility execution point of view, temporal o.c plan may be just enough. The temporal o.c plan can be built from a logical o.c plan by sorting the logical relations between each paif of actions. It's not clear how to build a logical o.c plan from a temporal o.c plan, though.

[9]lp_solve can be downloaded from http://www.cs.wustl.edu/~javagrp/help/LinearProgramming.html

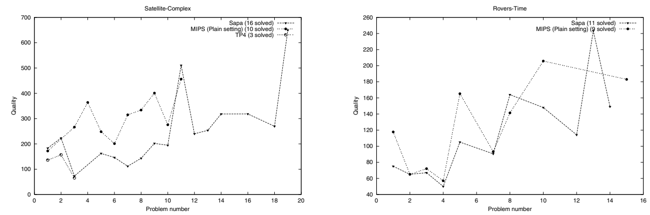


Figure 3: Comparing the quality (in terms of makespan) of the plan returned by *Sapa* to MIPS and TP4 in "Satellite" and "Rover" domains–two of the most expressive domains motivated by NASA applications.

Evaluating the Effect of Greedy Partialization

The first test suite is the 80 random temporal logistics provided with the TP4 planner. In this planning domain, trucks move packages between locations inside a city and airplanes move them between cities. Figure 2 shows the comparison results for only the 20 largest problems, in terms of number of cities and packages, among 80 of that suite. In the left graph of Figure 2, trucks are allowed to move packages between different locations in different cities, while in the right graph of the same figure, 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 (Smith & Weld 1999)); (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 (Refanidis & Vlahavas 2001), or MIPS (Edelkamp 2001) 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 plan for the problem (as produced by TGP and TP4), 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* are on the average 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 inter-city driving 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.[10]

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 (Refanidis & Vlahavas 2001) or MIPS (Edelkamp 2001) and in this case, the improvement would be 40% and 45% respectively for the two problem sets.

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

domains	orig/tt-dur	gpp/tt-dur	gpp/orig
zeno simpletime	0.8763	0.7056	0.8020
zeno time	0.9335	0.6376	0.6758
driverlog simpletime	0.8685	0.5779	0.6634
driverlog time	0.8947	0.6431	0.7226
satellite time	0.7718	0.6200	0.7991
satellite complex	0.7641	0.6109	0.7969
rovers simpletime	0.8204	0.6780	0.8342
rovers time	0.8143	0.7570	0.9227

Table 1: Comparison of different makespan values in the IPC's domains

domains	#Solved	Diff GP	Aveg. Diff
zeno simpletime	8/13	2/2	0.9723
zeno time	10/10	0/0	1.0
driverlog simpletime	11/12	2/2	0.9748
driverlog time	10/12	1/2	0.9928
satellite time	14/15	0/3	1.0
satellite complex	12/12	0/1	1.0
rovers simpletime	4/11	2/2	0.9276
rovers time	3/9	2/3	0.8355

Table 2: Comparison of optimal and greedy partializations

Use of partialization at IPC-2002

The greedy partialization technique described in this paper was part of the implementation of *Sapa* with which we took part in the International Planning Competition (IPC-2002). At IPC, *Sapa* was one of the best planners in the most expressive metric temporal domains, both in terms of planning time, and in terms of plan quality (measured in makespan). The credit for the plan quality can be attributed in large part to the partialization technique. In Figure 3, we show the comparison results on the quality of plans returned by *Sapa* and its nearest competitors from the Satellite (complex setting) and Rovers domains–two of the most expressive domains at IPC, motivated by NASA applications.[11] It is interesting to note that although TP4 (Haslum & Geffner 2001) guarantees optimal makespan, it was unable to solve more than 3 problems in the Satellite domain. *Sapa* was able to leverage its search in the space of position-constrained plans to improve search time, while at the same time using post-processing to provide good quality plans.

Figures 4 provides more detailed comparison of the makespan values before partialization, after greedy partialization, and after optimal partialization. We use problems of the four domains used in the competition, which are: ZenoTravel, DriverLog, Satellite, and Rovers. For each domain, we use two sets of problems of highest levels, and take the first 15 (among the total of 20) problems for testing. The *simple-time* sets involve durative actions without resources, and the *time/complex* sets (except the DriverLog domain) involve durative actions using resources. In each of the four figures, we show the comparison between the makespans of a (i) serial plan, (ii) a parallel p.c plan returned by *Sapa* (iii) an o.c plan built by greedy partialization, and (iv) an o.c plan returned by solving the MILP encoding. Because the two optimal-makespan planners that participated in the competition—TP4 and TPSYS—could only solve the first few problems in each domain, we could not include the optimal makespan values in each graph.

For this set of problems, we discuss the effect of greedy postprocessing here and leave the comparison regarding the results of optimal postprocessing until the next section. Table 1 summarizes the comparison between different makespan values for 8 sets of problems in those 4 domains. The three columns show the fractions between the makespans of greedily partialized o.c plan (gp), the original parallel p.c plan (orig), and the total duration of actions in the plan (tt-dur), which is equal to the makespan of a serial plan. Of particular interest is the last column which shows that the greedy partialization approach improves the makespan values of the original plans ranging from 8.7% in the RoversTime domain to as much as 33.7% in the DriverLog Simpletime domain. Compared to the serial plans, the greedily partialized o.c plans improved the makespan values 24.7%-42.2%.

The cpu times for greedy partialization are extremely short. Specifically, they were less than 0.1 seconds for all problems with the number of actions ranging from 1 to 68. 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 (Nguyen & Kambhampati 2001), 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.

Optimal Makespan Partialization

We would now like to empirically characterize the how far the makespan of the plan produced by greedy partialization is in comparison to that given by optimal parallelization. To compute the optimal parallelization, we use the MILP encoding discussed earlier and solve them using the Java version of LP_SOLVE, a public domain integer programming solver.

Table 2 shows the statistics of solving the 8 sets of problems listed in Figures 4. The objective function is to minimize the makespan value. The first column shows the number of problem that can be solved by LP_SOLVE (it crashed when solving the other encodings). For example, for ZenoSimpletime domains, LP_SOLVE can solve 8 of 13 encodings. In the second column, we show the number of problems, among the ones solvable by LP_SOLVE, for which the optimal o.c plan is different from the greedily partial-

[11] The competition results were collected and distributed by the IPC3's organizers and can be found at (Fox & Long 2002). Detailed descriptions of domains used in the competition are also available at the same place.

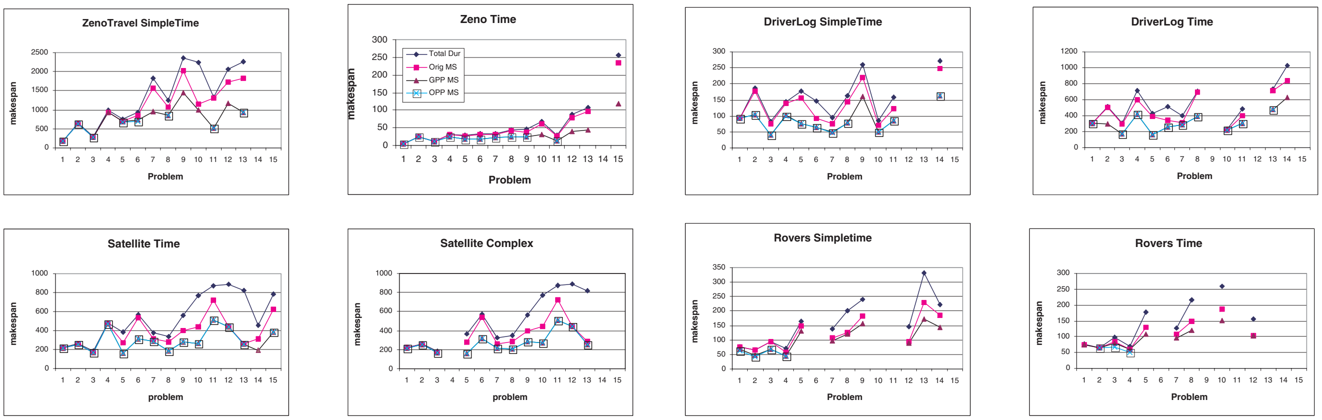


Figure 4: Comparison of different makespan values for problems in Zeno-Simpletime, Zeno-Time, DriverLog-SimpleTime, DriverLog-Time, Satellite-Time, Satellite-Complex, Rovers-Simpletime, and Rovers-Time domains.

ized o.c plan. For example, in the RoversTime domains, there are 3 problems in which the two o.c plans are different and in 2 of them, the optimal makespans are smaller than the greedily partialized makespans. The third column shows the average ratio between the optimal and greedy makespans (averaged over only those problems for which they are different). The table shows that in the ZenoTime domain, for all 10 problems, the optimal o.c plans and greedy o.c plans are identical. For the two Satellite domains (time and complex) there are 4 problems in which the two o.c plans are different, but there is no makespan improvement in any of these cases. In the ZenoSimpletime and two Driverlog domains, there are few problems in which the two o.c plans are different but the makespan improvements are very small (0.7-2.8%). The most promising domains for the optimal partialization approach is the Rovers domains in which 2 of 4 solved problems in RoversSimpletime and 2 of 3 solved problems in RoversTime have better optimal o.c plans than greedy o.c plans. The improvements range from 7.3% in RoversSimpletime to 16.4% in RoversTime. For the Rovers domain, the encodings seem to be more complicated than other domains and LP_SOLVE crashed in solving most of them. In terms of the time for solving the encodings, where LP_SOLVE was able to solve the encodings, it did so within several seconds. As we mentioned, it did "crash" on several larger encodings. We are planning to use more powerful and robust solvers such as CPLEX to find the solutions for the remaining unsolved encodings.

The results in Table 2 clearly show that optimal partialization is often not very much better than the greedy partialization. This is surprising, considering the fact that the former allows for reordering of actions. (i.e., the partialization does not have to be consistent with the orderings in the p.c plan). We believe that these results can be explained by looking at the potential number of supports for the causal link constraints (i.e. number of supporters for a given action's precondition) in the p.c plans output by *Sapa* in the various domains. It turns out that in all domains other than Rovers, the average number of supports is very close to 1. For example, in the Satellite domains, the largest number of supporters for a causal link is 2, but 93% of causal links has single support. In the ZenoTravel and DriverLog domains, the percentage of causal links with single support is lesser, but the largest number of supporters for a causal link is still only 4 (and most of the problems still only have causal links of 1 are 2 supports). Because of the dominance of causal links with single supports, there are only few candidate o.c plans and thus it is more likely that the greedy and optimal partialization routines will output the same o.c plans. In the Rovers domain, there are causal links with upto 12 supporters, and there is no problem in which the causal link have only 1 or 2 supporters. The percentage of single support constraints is only 77.8% for RoversSimpletime and 81.3% for RoversTime domains. Because there are more causal links with more supporters, there are more potential o.c plans for a given set of actions. Thus, there is more chance that the optimal partialization will find a better o.c plan than the greedy partialization approach.

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 that attempt to improve the temporal flexibility of plans was the work by Fade and Regnier (Fade & Regnier 1990) who discussed an approach for removing redundant orderings from the plans generated by STRIPS system. Later work by Mooney (Mooney 1998) and Kambhampati and Kedar (Kambhampati & Kedar 1994)characterized this partialization process as one of explanation-based order generalization. Backstrom (Backstrom 1998) 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 (Veloso et. al. 1990) and (Kambhampati & Kedar 1994) can be seen as providing a greedy value ordering strategy over the partialization encoding for classical plans. However, unlike the greedy strategies presented in this paper, 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" (Kautz et. al. 1996). 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 (Mali 1999). 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 (Kautz et. al. 1996) are limited to classical domains.

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. Currently, we are focusing on (i) improving the optimal solving of MILP encodings by finding better solver; (ii) testing with different objective functions other than minimize makespan; (iii) developing greedy value ordering strategies that are sensitive to other types of temporal flexibility measures besides makespan; and finally our ultimate goal is (iv) building a stand-alone partialization software (separate from *Sapa*) that can take any p.c/o.c plan returned by any planner and greedily or optimally partialize it.

References

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

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

Bonet, B., Loerincs, G., and Geffner, H. 1997. A robust and fast action selection mechanism for planning. *Proc AAAI-97*

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

Do, M., and Kambhampati, S. 2002. Planning graph-based heuristics for cost-sensitive temporal planning. *In Proc. AIPS-02*.

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

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

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*

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

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

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

Hoffmann, J. 2000. http://www.informatik.uni-freiburg.de/ hoffmann/ff.html

ILOG Solver Suite. http://www.ilog.com/products/solver/

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

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

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.*

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

Mali, A. Plan Merging & Plan Reuse as SAT. *Proc ECP-99.*

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

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

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 AIJ.*

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

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

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

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

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.*

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

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

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