

# Planning with Disjunctive Temporal Constraints

Peter J. Schwartz and Martha E. Pollack

Department of Electrical Engineering and Computer Science  
University of Michigan  
Ann Arbor, MI, USA  
pschwart@eecs.umich.edu, pollackm@eecs.umich.edu

## Abstract

Many real-world problems contain elements of both planning and scheduling problems. In response, many frameworks and systems have been developed that add scheduling abilities to planning systems. We continue this trend by presenting a framework and algorithm for planning with disjunctive temporal constraints. We define a constraint-based interval (CBI) framework for describing a planning problem with disjunctive temporal constraints, which we call a disjunctive temporal planning problem (DTPP). We also describe the Disjunctive Temporal Partial-Order Planner (DT-POP), a planning algorithm that is capable of solving DTPP's, and analyze several techniques for improving its efficiency. Although the current implementation of DT-POP is not yet competitive with other planning systems in terms of efficiency, it is more expressive in terms of complex temporal constraints. We offer several ideas for improving the efficiency of DT-POP in the future, in hopes that the current work will be the beginning of an interesting line of research.

## Introduction

Several recent papers (e.g., Smith, Frank, & Jónsson 2000, Boddy 2003) have pointed out that many real-world problems require a combination of planning techniques and scheduling techniques. If we are to design a planning system that incorporates scheduling techniques in order to solve such problems, then the planning system must accommodate a rich representation for temporal constraints. A temporal planner must account for the fact that actions take time, with conditions and effects occurring at various times throughout an action's duration. It must also account for the fact that exogenous events can change the environment and that goals are sometimes subject to temporal constraints.

In the past decade, several temporal frameworks have been proposed to represent such temporal relationships (e.g., Ghallab & Laruelle 1994, Muscettola 1994, Smith & Weld 1999, Fox & Long 2003), with varying levels of expressive power, computational cost, and ease of use. The use of disjunctive temporal constraints to represent

temporal relationships in planning problems has so far been avoided because, in general, solving a disjunctive temporal problem (DTP) is NP-hard (Dechter, Mieri, & Pearl 1991). Of course, if an action does include disjunctive temporal constraints, it would be possible to create a separate copy of the action for each possible combination of simple temporal constraints. If an action contains multiple disjunctions, however, this approach could lead to an exponential increase in the number of actions. In addition, this approach ignores the fact that the simplified copies of the original disjunctive action would share many of the same temporal constraints, missing an opportunity to improve efficiency. If a planning problem also contains disjunctive temporal constraints on the goals and exogenous events, expanding all possible combinations of constraints would essentially create a potentially exponential number of planning problems.

Recent advances in DTP solvers (Stergiou & Koubarakis 1998, Møller & Lichtenberg 1998, Armando, et al. 1999, Oddi & Cesta 2000, Tsamardinos & Pollack 2003) have led to significant efficiency gains in DTP checking. With these advances, it may now be feasible to use disjunctive temporal constraints as a common framework to represent all temporal constraints on actions, exogenous events, and goals within a temporal planning problem. We define such a framework in the next section and illustrate it with an example. We then present the algorithm for the Disjunctive Temporal Partial-Order Planner (DT-POP), which is capable of solving planning problems with disjunctive temporal constraints. Although the current implementation of DT-POP is not yet competitive with other planning systems in terms of efficiency, it is more expressive in terms of complex temporal constraints. We offer several ideas for improving the efficiency of DT-POP in the future, in hopes that the current work will be the beginning of an interesting line of research.

## The Disjunctive Temporal Planning Problem

We build on the constraint-based interval (CBI) formalism (Smith, Frank, & Jónsson 2000) by adding disjunctive temporal constraints to describe temporal relationships over preconditions, effects, exogenous events, and goals. We call this extension the disjunctive temporal planning

problem (DTPP). We begin by defining the basic components.

**Definition 1:** A **timepoint** is a real-valued variable representing a 0-duration moment in time.

As in all CBI formalisms, we model conditions as extending over an interval delimited by a pair of timepoints.

**Definition 2:** A **condition** is a triple  $\langle tp_1, L, tp_2 \rangle$ , where  $tp_1$  and  $tp_2$  are timepoints such that  $tp_1 \leq tp_2$  and  $L$  is a set of literals.

A condition  $\langle tp_1, L, tp_2 \rangle$  means that each literal of  $L$  holds true without interruption over the time interval  $[tp_1, tp_2]$ . Conditions are used to model preconditions and effects of actions, as well as goals and exogenous events. The interval identified in a condition is the *minimal* period over which the literals in the condition are guaranteed to hold. We make a persistence assumption that each literal of  $L$  continues to hold after the time of  $tp_2$  until it is negated by some other condition. We describe the temporal relationships between the timepoints of conditions with disjunctive temporal constraints.

**Definition 3:** A **disjunctive temporal constraint** (Stergiou & Koubarakis 1998) is a disjunction of simple temporal constraints. A **simple temporal constraint** (Dechter, Mieri, & Pearl 1991) is an inequality of the form  $tp_2 - tp_1 \leq b$  where  $tp_1$  and  $tp_2$  are timepoints and  $b$  is a real number. Intuitively, such a constraint is satisfied if  $tp_2$  follows  $tp_1$  by no more than  $b$  units of time. A disjunctive temporal constraint is satisfied if at least one of its component simple temporal constraints is satisfied. A **schedule** is an assignment of timepoints to real values.

Disjunctive temporal constraints are able to model a wide variety of temporal relations, including both qualitative ones (equality, ordering, and, notably, inequality) and quantitative ones (durations). They can also represent any combination of conjunctions and disjunctions of simple temporal constraints. Consequently, in the remainder of the paper, we will write constraints in the most natural form, without converting them to DTP form.

We can now combine these components into a single structure that is able to model actions whose preconditions and effects hold at arbitrary times and intervals.

**Definition 4:** A **disjunctive temporal action (dt-action)** is a triple  $\langle Pre, Eff, D \rangle$ , where  $Pre$  (the preconditions) and  $Eff$  (the effects) are sets of conditions, and  $D$  is a set of disjunctive temporal constraints over the timepoints in  $Pre$  and  $Eff$ .

Semantically, a dt-action  $\langle Pre, Eff, D \rangle$  means that if all of the preconditions of  $Pre$  hold at times consistent with  $D$

and the action is performed, then it will result in all of the effects of  $Eff$  at times consistent with  $D$ .

**Definition 5:** A **disjunctive temporal planning problem (DTPP)** is a 4-tuple  $\langle G, E, D, A \rangle$ , where  $G$  (the goals) and  $E$  (the exogenous events) are sets of conditions,  $D$  is a set of disjunctive temporal constraints over the timepoints in  $G$  and  $E$ , and  $A$  is the set of available dt-actions.

Notice that the temporal constraints in  $D$  are over the timepoints in  $G$  and  $E$ , meaning that they describe the temporal relationships among goals and exogenous events. These are separate from the temporal constraints within each action of  $A$ .

**Definition 7:** A **disjunctive temporal plan (dt-plan)** is a triple  $\langle Need, Have, D \rangle$ , where  $Need$  is a set of conditions representing the goals and action preconditions,  $Have$  is a set of conditions representing the exogenous events and action effects, and  $D$  is a set of disjunctive temporal constraints over the timepoints in  $Need$  and  $Have$ .

A solution to a DTPP is a dt-plan in which we “have” achieved everything we “need” when we need it.

**Definition 8:** A dt-plan  $\langle Need, Have, D \rangle$  is a **solution** to a DTPP provided that: 1)  $D$  is consistent, and, in any schedule of the timepoints in  $Need$  and  $Have$  that is consistent with  $D$ , 2) no conditions of  $Have$  with opposing literals overlap, and 3) for every literal  $l \in nL$  of each  $Need$  condition  $\langle ntp_1, nL, ntp_2 \rangle$ , there exists a  $Have$  condition  $\langle htp_1, hL, htp_2 \rangle$  such that  $l \in hL$ ,  $htp_1 \leq ntp_1$ , and  $ntp_2 \leq htp_2$ . These conditions are the DTPP analogues of consistency (condition 1), no threats (condition 2), and establishment of all open conditions (condition 3).

## Example

We will now present a simple example of a temporal planning problem to demonstrate a DTPP. The Wonderful Waterproof Widget Company plans to offer a new service in which it guarantees to produce any order of up to three custom widgets in one hour or less. The company can offer this service thanks to the Widget Wizard, a new automated machine that makes widgets. A widget is a simple device that consists of two parts, a left half and a right half. A customer may choose to paint each half either green or blue by pushing a few buttons on the Widget Wizard.

After a customer places an order, the Widget Wizard produces each widget using three basic steps: 1) it paints both halves according to the customer’s order, 2) joins the halves by gluing or screwing them together, and 3) seals the widget to make it waterproof. The Widget Wizard has unlimited paint, glue, screws, and sealant at its disposal, but it only has one screwdriver and one brush. To paint part of a widget, the Widget Wizard must first pry open a can of paint with the screwdriver for 1 minute, then use the

brush to paint the part for 5 minutes, and then let the part dry for 5 minutes. The Widget Wizard contains only four cans of paint—two blue and two green. Every 20 minutes, each paint can is automatically refilled for 1 minute and then closed, regardless of whether the paint cans are open, closed, full, or empty.

To glue two widget halves together, the Widget Wizard must apply the glue for 2 minutes and then let it dry for 7 minutes. Alternatively, the Widget Wizard can screw two widget halves together by using the screwdriver for 6 minutes. Finally, to seal a widget, the Widget Wizard must dip it into two separate chemical sealants. The widget must be dipped in either sealant for at least 4 minutes, and must be dipped in the other sealant for at least 1 minute. Furthermore, the widget must dry for 3 to 6 minutes between dips. Only one widget at a time can be dipped in each sealant.

Due to space limitations, we will present the encoding of only part of the Wizard Widget domain. The initial state does not have a duration associated with it, so we can represent it with a single condition:

$$C_I = \{\langle init, \text{InitialState}, init \rangle\}$$

where *init* is a timepoint and InitialState is a label that is used as shorthand to denote the set of literals that state that the tools are clean and available and that none of the paint cans are open.

The other exogenous events of the widget example are the automatic refilling and closing of all of the paint cans. Since the paint cans are refilled every 20 minutes and all of the widgets must be finished in 60 minutes or less, the problem only needs to specify two automatic refills. Each is represented by two conditions—one to state what is true during the refill, and the other to state what is true afterwards—and two simple temporal constraints—one to specify the start time of the refill, and the other to specify the duration:

$$C_R = \{\langle refill1B, \text{DuringRefill}, refill1E \rangle, \\ \langle refill1E, \text{AfterRefill}, refill1E \rangle, \\ \langle refill2B, \text{DuringRefill}, refill2E \rangle, \\ \langle refill2E, \text{AfterRefill}, refill2E \rangle\}$$

$$D_R = (refill1B - init = 20) \wedge (refill1E - refill1B = 1) \wedge \\ (refill2B - init \geq 40) \wedge (refill2E - refill2B = 1)$$

where DuringRefill is a label for the set of literals that states that each can of paint is unavailable while being refilled, and, similarly, AfterRefill states that each can of paint is full, closed, and available after being refilled.

The Widget Wizard example has a relatively simple goal specification in which a single set of goal literals is constrained to be achieved by a deadline, and so it can be represented with a single condition and a single temporal constraint:

$$C_G = \{\langle goal, \text{GoalConditions}, goal \rangle\}$$

$$D_G = (goal - init \leq 60)$$

where GoalConditions states that all of the widgets are painted, joined, and sealed. The exact set of literals contained in GoalConditions would depend on the number and color of widgets that a user requests from the Widget Wizard. In general, a DTPP can represent multiple sets of goal literals, where each set is constrained to occur at different times or to hold for different durations of time relative to any exogenous event or any other goal.

To illustrate the expressiveness of the DTPP framework, we will define only the most complex action of this problem, the Seal action. It includes dipping the widget into one sealant, letting it dry, and then dipping it into the other sealant. Each sealant is a unary resource that is reserved only over the sub-interval of time that it is needed within the action, not over the entire duration of the action. This makes it possible to overlap actions more tightly. The Seal action also contains a delayed effect (letting the widget dry), which we can easily represent in a dt-action with an extra effect that is constrained to a different time. The other temporal relationships that describe the Seal action contain inequalities and disjunctions, which can be expressed with the disjunctive temporal constraints of a dt-action. These temporal constraints bind the parts of the Seal action together, so we represent the Seal action as one large dt-action rather than splitting it up into smaller sub-actions:

$$Pre = \{\langle seal1B, \{\text{Painted}(\text{left}, \text{color1}), \\ \text{Painted}(\text{right}, \text{color2}), \text{Joined}(\text{left}, \text{right})\}, seal2E \rangle, \\ \langle seal1B, \{\text{Available}(\text{seal1})\}, seal1B \rangle, \\ \langle seal2B, \{\text{Available}(\text{seal2})\}, seal2B \rangle\}$$

$$Eff = \{\langle seal1B, \{\sim \text{Available}(\text{seal1})\}, seal1E \rangle, \\ \langle seal1E, \{\text{Available}(\text{seal1})\}, seal1E \rangle, \\ \langle seal2B, \{\sim \text{Available}(\text{seal2})\}, seal2E \rangle, \\ \langle seal2E, \{\text{Sealed}(\text{left}, \text{right}), \text{Available}(\text{seal1})\}, seal2E \rangle\}$$

$$D = ((seal1E - seal1B \geq 1 \wedge seal2E - seal2B \geq 4) \vee \\ (seal1E - seal1B \geq 4 \wedge seal2E - seal2B \geq 1)) \wedge \\ (seal2B - seal1E \geq 3) \wedge (seal2B - seal1E \leq 6)$$

Note that the timepoint *seal2B* is constrained to occur in the time interval of 3 to 6 minutes after the timepoint *seal1E*. If the temporal constraints on actions, goals, or exogenous events do not restrict a timepoint to a single value, it means that the planning system is free to schedule that timepoint to any time that is consistent with the temporal constraints; it does not mean that the exact timing is uncertain and the planning system is forced to accommodate that uncertainty. Clearly, it would be desirable to explicitly model such temporal uncertainty: the STPu formalism (Morris, Muscettola, & Vidal 2001) provides this capability, and we will explore its use in subsequent work.

## The DT-POP Algorithm

Solving a DTPP is challenging, because the a set of disjunctive temporal constraints may not impose a total ordering of timepoints, and consequently we cannot apply many of the recent, highly efficient algorithms that rely on a total ordering (e.g., forward-search-based planning (Hoffmann & Nebel, 2000) or planning-graph-based planning (Long & Fox, 1999)). Instead, we adopt a partial-order planning (POP) approach, using a standard algorithm as given in Figure 1.

```

DT-POP(DTPP ⟨G, E, D, A⟩)
  P0 = InitialPlan(G, E, D)
  Q = InitializeQueue(P0)
  while not Empty(Q)
    P = Pop(Q)
    if Satisfiable(TemporalConstraints(P)), then
      if Solution(P), return P
      Children = Expand(P, A)
      for each Child plan in Children
        Insert(Child, Q)
  return FAIL

```

**Figure 1.** The DT-POP algorithm.

DT-POP searches a space of dt-plans, starting from the initial dt-plan that describes the goals (the *Need* conditions of the initial plan) and exogenous events (the *Have* conditions of the initial plan) and expanding the search space by repairing flaws. In a dt-plan  $\langle \text{Need}, \text{Have}, D \rangle$ , an open condition is any literal  $l \in nL$  of a *Need* condition  $\langle ntp_1, nL, ntp_2 \rangle$  for which there exists no *Have* condition  $\langle htp_1, hL, htp_2 \rangle$  such that  $l \in hL$ ,  $htp_1 \leq ntp_1$ , and  $ntp_2 \leq htp_2$ . A threat is any pair of *Have* conditions with opposing literals that can possibly overlap in a schedule that is consistent with  $D$ .

The first major difference between DT-POP and other POP algorithms occurs during the addition of actions to a dt-plan to achieve open conditions. Actions and plans have the same structure in the DTPP formalism, so adding a dt-action to a dt-plan is more like merging a sub-plan into an existing plan rather than adding a single step. Although this requires more work on the part of the planner than adding, say, a STRIPS operator or a durative action, this is what gives DT-POP its ability to represent complex actions. As with most POP algorithms, DT-POP employs a least commitment strategy and leaves variables introduced by the addition of new actions unbound as long as possible to reduce the branching factor.

DT-POP achieves open conditions in a process that is analogous to adding a causal link in classical POP planning. First, DT-POP finds an effect or exogenous event that establishes the literal of the open condition. If DT-POP achieves the open condition with a new dt-action, it must add all of the action’s preconditions, effects, and temporal constraints. Preconditions are added to the *Need*

conditions, and effects are added to the *Have* conditions. Note that we cannot just push back the end time of the *Have* condition that achieves the literal because this would commit to protecting *all* of the literals of the *Have* condition instead of just the one that is needed. Instead, DT-POP adds a new *Have* condition that protects the needed literal until the end of the *Need* condition, along with temporal constraints so that the new *Have* condition contains the *Need* condition. If a dt-action includes multiple effects that can achieve the open condition, then one child dt-plan is created for each such effect.

The next difference between DT-POP and classical POP algorithms is the way in which threats are resolved. Traditionally, POP algorithms resolve threats by separation, promotion, or demotion. In DT-POP, a threat is resolved either by separation, or by adding a disjunctive temporal constraint that forces the end of one *Have* condition to precede the beginning of the other, or visa versa; that is, the new disjunctive temporal constraint delays the choice between promotion and demotion until the plan’s temporal constraints are checked by the DTP-solver. This is equivalent to the disjunctive ordering constraints that RePOP (Nguyen & Kambhampati 2001) uses to resolve threats. Essentially, this eliminates the need to create two child plans during threat resolution, but at the cost of requiring the DTP-solver to manage the newly added constraint.

The last difference between DT-POP and classical POP algorithms is the checking of temporal constraints. The temporal constraints of a dt-plan are checked for consistency just before it is expanded. Classical POP algorithms only represent ordering constraints, so consistency checking only takes polynomial time. DT-POP represents disjunctive temporal constraints, so consistency checking is NP-hard (Dechter, Mieri, & Pearl 1991). Even though DT-POP uses an efficient DTP-solver, each consistency check can be time-consuming. In the next section we explore several techniques to speed up the process.

## Efficiency Techniques

Each call to a DTP-solver can be very expensive. To help handle this complexity, we used an efficient DTP-solver, Epilitis (Tsamardinos & Pollack 2003). We were also able to identify several techniques to improve the efficiency of DT-POP by getting the most out of each call to Epilitis. These techniques include delayed temporal constraint checking, conflict-directed backjumping, and no-good recording.

The first efficiency technique is to delay the temporal constraint checking. In a sense, a call to the DTP-solver is only helpful either when it finds that the temporal constraints of a dt-plan are unsatisfiable (so it does not waste time expanding it), or when it finds that the temporal constraints of a dt-plan with no open conditions or threats is satisfiable (so it can return it as a solution). Hence, each call to Epilitis on a dt-plan that has satisfiable temporal

constraints but still contains flaws to repair wastes time. Rather than calling the DTP-solver on every single dt-plan before it is expanded, DT-POP might be sped up by deferring this temporal constraint check until a dt-plan has been modified several times. Since the search space of DT-POP is a tree (with the initial plan as its root), DT-POP can accomplish this by calling the DTP-solver on a plan after a fixed number of levels have passed since its last ancestor plan was checked. We tested this hypothesis by deferring the temporal constraint check until after 1 level (i.e., every dt-plan was checked before expansion), 5 levels, and 10 levels. We also deferred the temporal constraint check until the very end after all open conditions and threats were resolved.

The other two efficiency techniques are based on a common idea: prune as many plans as possible after each call to the DTP-solver by exploiting the fact that many dt-plans have temporal constraints in common. If the DTP-solver Epilitis is invoked and determines that a plan's temporal constraints are unsatisfiable, it returns a justification to DT-POP. A justification is a minimal subset of temporal constraints that together are unsatisfiable, so any set of temporal constraints that is a superset of a justification is also unsatisfiable. This means that once Epilitis has identified a problem with the temporal constraints of one plan, DT-POP can prune other plans with the same problem.

For example, say DT-POP is planning to fill an order of widgets. Assume that DT-POP is deferring the temporal constraint check until the end. It is possible that DT-POP will generate a dt-plan that tries to paint four widget parts before the first automatic refill, but it is actually only possible to paint three. At this point, the plan contains unsatisfiable temporal constraints, but DT-POP is unaware of this because it has not checked them in Epilitis yet. If the plan has open conditions or threats, DT-POP will repair all of these flaws before calling Epilitis. Since temporal constraints can only be added to a plan as flaws are repaired, every plan in this subtree of the search space contains the same subset of unsatisfiable temporal constraints. By exploiting the justification returned from Epilitis, DT-POP could quickly recognize that all of these plans contain the same problem.

Our second efficiency technique exploits justifications by performing conflict-directed backjumping (CDB), an idea that originated in research on constraint satisfaction problems (CSP's) (Prosser 1993). DT-POP performs CDB by finding the earliest ancestor of the plan that contains all of the temporal constraints in the justification. This would be the plan in which DT-POP first decided to try to paint four widget parts before the first automatic refill. Temporal constraints can only be added to a plan when it is expanded, so all of the descendants of the earliest ancestor must also contain the temporal constraints in the justification and must therefore also be unsatisfiable, so DT-POP deletes these plans from the search queue. CDB requires a small amount of extra space to store parent and child pointers in each plan.

The final efficiency technique, which also exploits justifications, is no-good recording (NGR), which also originated in CSP research (Schiex & Verfaillie 1994). DT-POP performs NGR by storing each new justification and then checking each plan against the set of stored justifications just before expanding it. In some cases, nearby plans in the search space share a subset of exactly the same temporal constraints, even though no common ancestor plan contains that same subset. In the widget example, DT-POP might create two branches in the search space by achieving an open condition in two different ways. Each branch might then lead to a plan in which the Widget Wizard tries to paint four widget halves before the first refill. When Epilitis checks the temporal constraints of the first such plan and returns a justification, CDB would not delete the other plan because they do not share a common ancestor with all of the constraints from the justification. NGR would store the justification from the first call to Epilitis and, seeing that the second plan contains the same problematic constraints, delete it without making a second call to Epilitis. NGR can consume a fair amount of extra space to store each justification.

We tested each of these three techniques (and all combinations of them) on two sets of random DTPP's. The first set contained 8 actions, each with 1 to 3 preconditions, 1 to 3 effects, and 0 to 2 temporal constraints. The second set was slightly more complex, containing 9 actions, each with 1 to 3 preconditions, 1 to 4 effects, and 1 to 2 temporal constraints. For each set of dt-actions, we created five random problems. Each problem contained 10 to 20 initial conditions, 1 to 3 goals, and 1 to 2 exogenous events. Each goal and exogenous event contained 1 to 2 literals. The temporal constraints were created by randomly selecting begin and end times for all of the exogenous events and 1 to 2 of the goals, relative to the initial state. Because our implementation makes use of random processes when ordering plans in the search queue, we ran each problem three times on each combination of efficiency techniques and report the averages across all three runs on all ten problems. Each run was limited to 300 seconds. DT-POP was implemented and compiled in Allegro Common LISP 6.2, and run on a Windows-based machine with a 3.0 GHz Pentium 4 processor with 512MB of memory. The results of our experiments are shown in Figure 2.

The first row of Figure 2 shows the performance of DT-POP without using any of the three efficiency techniques. In this row, the time spent checking temporal constraints in Epilitis accounts for 86% of the total run time of DT-POP, which means our efficiency techniques are aimed at the root of the problem. In all of the rows, the time spent checking temporal constraints in Epilitis accounts for all but about 10 seconds of the total run time of DT-POP. Even though our efficiency techniques had a significant impact on time spent checking temporal constraints, they had no adverse impact on the rest of the planning processes performed by DT-POP.

| Level | CDB | NGR | Total Time | DTP Time | DTP Calls | Del   | Gen    | Exp    | % Comp |
|-------|-----|-----|------------|----------|-----------|-------|--------|--------|--------|
| 1     | N   | N   | 86.83      | 74.75    | 84.50     | 0.00  | 101.67 | 79.97  | 80.00  |
| 5     | N   | N   | 101.42     | 91.61    | 821.40    | 0.00  | 895.97 | 877.90 | 70.00  |
| 10    | N   | N   | 92.20      | 81.68    | 799.27    | 0.00  | 882.67 | 863.17 | 73.30  |
| end   | N   | N   | 128.11     | 117.99   | 904.00    | 0.00  | 965.10 | 950.23 | 60.00  |
| 1     | Y   | N   | 92.73      | 80.78    | 85.07     | 4.07  | 105.90 | 80.20  | 80.00  |
| 5     | Y   | N   | 71.06      | 59.02    | 58.47     | 23.07 | 242.23 | 185.47 | 86.70  |
| 10    | Y   | N   | 49.24      | 38.03    | 49.03     | 33.80 | 292.67 | 222.83 | 93.30  |
| end   | Y   | N   | 31.05      | 19.91    | 36.67     | 49.67 | 406.07 | 308.97 | 96.70  |
| 1     | N   | Y   | 92.76      | 81.18    | 80.47     | 11.57 | 108.30 | 87.00  | 80.00  |
| 5     | N   | Y   | 57.88      | 46.53    | 38.50     | 40.77 | 237.30 | 204.63 | 96.70  |
| 10    | N   | Y   | 32.79      | 21.57    | 19.17     | 51.73 | 260.47 | 226.50 | 100.00 |
| end   | N   | Y   | 12.26      | 1.11     | 2.13      | 89.30 | 506.90 | 458.47 | 96.70  |
| 1     | Y   | Y   | 87.84      | 75.93    | 75.80     | 12.07 | 103.00 | 77.40  | 80.00  |
| 5     | Y   | Y   | 61.99      | 51.02    | 42.70     | 54.90 | 268.37 | 211.37 | 93.30  |
| 10    | Y   | Y   | 35.72      | 24.40    | 18.53     | 44.77 | 254.67 | 201.57 | 96.70  |
| end   | Y   | Y   | 12.41      | 1.06     | 1.97      | 53.03 | 332.60 | 269.20 | 100.00 |

**Figure 2.** Effects of efficiency techniques on DT-POP.

Columns indicate the following: Level = levels passed between DTP calls, CDB = whether CDB was used, NGR = whether NGR was used, Total Time = total time, DTP Time = time spent in DTP-solver, DTP Calls = number of calls to DTP-solver, Del = number of plans deleted by CDB and NGR, Gen = number of plans generated, Exp = number of plans expanded, % Comp = percentage of problems completed within 300 seconds

The first four rows in Figure 2 show us that without CDB or NGR, performance tends to degrade as temporal constraint checking was delayed further. This is reflected in the fact that the average total time per problem tends to increase while the percentage of problems completed within the time limit tends to decrease; however, there is clearly some fluctuation, and additional experiments would be warranted if not for the influence of the pruning strategies shown in the rest of the table. They show that when DT-POP uses CDB and/or NGR, delayed temporal constraint checking *helps* performance dramatically. Without CDB or NGR, early temporal constraint checking can catch problems sooner and prune plans before they are expanded further; delayed constraint checking allows a problem to propagate to more plans that must all be checked individually. With CDB or NGR, early temporal constraint checking wastes time on plans with satisfiable constraints; delayed temporal constraint checking still allows problems to propagate, but CDB and NGR prevent this from leading to multiple calls to Epilitis.

Our experiments revealed that CDB and NGR both made a significant improvement to the performance of DT-POP. As shown in Figure 2, DT-POP almost always spent less time per run when CDB or NGR was enabled. This reduction in time resulted from calling the DTP-solver far less frequently. With CDB or NGR, DT-POP was able to delete many plans from the search queue without ever calling Epilitis on them, leading to a solution after generating and expanding fewer plans, and ultimately

leading to the completion of more problems within the time limit.

Notice that NGR alone was more effective than CDB alone. NGR stores all of the justifications returned by Epilitis and compares them to all plans in the future. CDB uses each justification on a select subset of plans in the search space. Thus, the plans deleted by NGR is a superset of the plans deleted by CDB. This explains the fact that NGR consistently deletes more plans and makes fewer calls to Epilitis than does CDB. It also explains the fact that the performance of NGR with CDB is only slightly better than that of NGR without CDB. If we are willing to give up the space required to enable NGR, there is little use in also enabling CDB. It is possible that NGR will store so many justifications that the system runs out of space. Other research (Schiex & Verfaillie 1994) has explored methods for selectively keeping only the most useful justifications. If such a technique were employed, then NGR would not necessarily delete all of the plans that CDB would delete, so enabling CDB along with NGR might have a larger impact on performance.

## Related Work

Many other frameworks have been used to represent temporal constraints in planning problems. The PDDL 2.1 durative action representation (Fox & Long 2003) is one such framework, but this approach has its shortcomings. Smith (2003) points out that it is difficult to model

complex actions in which conditions are needed, or effects take place, at times or over intervals other than the start and end of the action. Smith offers an alternative temporal action representation in which conditions and effects can be defined over fixed intervals or at fixed points within an action. While this representation makes it possible to partially overlap actions with conflicting sub-intervals, it is limited in that the start and end times of these sub-intervals are fixed. Several systems (e.g., HSTS (Muscettola 1994), IxTeT (Ghallab & Laruelle 1994), EUROPA (Frank & Jónsson 2003)) represent simple temporal constraints on timepoints, allowing the start and end times of intervals to be scheduled at any time within a time window. The dt-actions of a DTPP also allow a problem to define action sub-intervals, but the disjunctive temporal constraints that describe their timing define *sets* of time windows for each sub-interval to start and end.

Long & Fox (2002) present a method for modeling exogenous events using PDDL 2.1 durative actions, but they admit that the technique “is not entirely elegant” [p.8]. PDDL 2.2 (Edelkamp & Hoffmann 2004) allows for timed initial literals that specify conditions that become true at specific times. Timed initial literals, however, are limited because they cannot represent literals that will continue to hold over an interval of time. Smith & Weld (1999) offer an extension of TGP that makes it possible to represent exogenous events and temporal constraints on goals. IxTeT can also take exogenous events as problem input. The DTPP framework uses disjunctive temporal constraints to represent the temporal relationships between goals and exogenous events, making it possible to describe deadlines on goals, time windows for achieving goals, maintenance goals, and temporally controllable exogenous events.

Planning graphs (Long & Fox 1999) represent a set of possible plans in a single structure. In a way, the use of disjunctive temporal constraints in DT-POP does the same thing. Each dt-plan in the search space can represent an exponential number of plans with simple temporal constraints. The disjunctive temporal constraints allow for a form of least commitment, leaving the choice of which disjunct to satisfy from each constraint up to the DTP-solver. Currently, temporal disjunctions are only added to a dt-plan to resolve threats. We could extend the use of disjunctive temporal constraints to represent the different ways of establishing open conditions. In this approach, all of the different ways of establishing an open condition could be represented in a disjunction in a single child node. It seems likely, however, that this would produce dt-plans with temporal constraints that are far too complex for any current DTP-solver to check in a reasonable amount of time.

## Discussion

We have defined the disjunctive temporal planning problem, which adds disjunctive temporal constraints to the CBI framework, making it possible to represent

planning problems with multiple goals at different times, exogenous events, resource intervals, and delayed effects. We have also described the DT-POP algorithm, and we have shown how it is able to solve DTPP’s. Finally, we have tested several techniques for improving the efficiency of the DT-POP algorithm, and we have shown that these techniques had a significant effect.

We still need to make major improvements to the efficiency of DT-POP before it is fast enough to be of practical use on large-scale planning problems. Solutions to our test problems generally required only 3 or 4 actions. DT-POP was able to solve only a handful of the simple time problems from the 3<sup>rd</sup> International Planning Competition (Long & Fox 2003) in less than 20 minutes. However, it should be noted that our goal to date in this work has been to develop a more expressive planning representation and begin to explore influences on its efficiency. We have not (yet) done extensive tuning of the code to promote maximal performance.

In addition to general code-tuning, we can cut down on the amount of time spent by the DTP-solver either by solving fewer DTP’s or by solving them faster. To solve fewer DTP’s, DT-POP could try to invoke the DTP-solver dynamically rather than at some fixed number of levels. The idea would be to look at the new temporal constraints that have been added to the plan since the constraints were last checked and only call the DTP-solver when it appears that the constraints have most likely become unsatisfiable. The difficulty is to find a method that can decide quickly and as accurately as possible when to call the DTP-solver without actually solving the DTP.

To solve DTP’s faster, DT-POP could make use of information from DTP’s that have already been solved. Currently, DT-POP treats each DTP that must be solved as if it were unrelated to other DTP’s encountered within the same planning problem (aside from CDB and NGR). In fact, each plan in the search space contains all of the temporal constraints contained by its parent plan, so the DTP’s are closely related. If DT-POP calls the DTP-solver on the temporal constraints of one plan and then on the constraints of its child, then it has repeated work and wasted time. DT-POP might be able to avoid this waste by storing extra information from the DTP-solver about the solution to the parent and passing that back to the DTP-solver when it checks the constraints of the child. Epilitis uses a meta-CSP approach to solve DTP’s, where each disjunctive constraint is a variable and each disjunct is a possible value. We might use a dynamic CSP (Schiex & Verfaillie 1994) approach in which the new constraints of the child plan correspond to new variables in the CSP. Instead of solving many separate DTP’s, the DTP-solver would solve one large dynamic DTP.

DT-POP is an initial attempt at planning with disjunctive temporal constraints. It represents the start of a potentially interesting line of research, but it remains to be seen whether or not planning with disjunctive temporal constraints can be made fast enough to be of any practical use. Although the DTPP framework extends the ability to represent certain temporal constraints, it does not offer any

representation of disjunctive preconditions, conditional effects, quantitative resources, continuous processes, or exogenous events that are triggered by conditions. These are important features in many real-world problems and would certainly make interesting extensions to the DTPP framework and the DT-POP algorithm.

## Acknowledgements

The material presented in this paper is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) through the Dept. of the Interior, NBC, Acquisition Services Division, under Contract NBCHD030010, and by the Air Force Office of Scientific Research (F49620-01-1-0066).

## References

- Armando, A., Castellini, C., and Giunchiglia, E. (1999). SAT-Based Procedures for Temporal Reasoning, In *Proc. 5th Eur. Conf. on Planning*, 97-108.
- Boddy, M.S. (2003). Imperfect Match: PDDL 2.1 and Real Applications. *Journal of Artificial Intelligence Research*, 20: 133-137.
- Dechter, R., Mieri, I., and Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, 49: 61-95.
- Edelkamp, S., and Hoffmann, J. (2004). PDDL2.2: The Language for the Classical Part of the 4<sup>th</sup> International Planning Competition.
- Fox, M., and Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20: 61-124.
- Frank, J., and Jónsson, A.K. (2003). Constraint Based Attribute and Interval Planning. *Journal of Constraints*, 8 no. 4: 339-364.
- Ghallab, M., and Laruelle, H. (1994). Representation and Control in IxTeT, a Temporal Planner. In *Proceedings 2nd International Conference on AI Planning Systems (AIPS-94)*, 61-67.
- Hoffmann, J., and Nebel, B. (2000). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14: 253-302.
- Joslin, D. (1996). Passive and Active Decision Postponement in Plan Generation. Ph.D. dissertation, Intelligent Systems Program, U. of Pittsburgh.
- Long, D., and Fox, M. (1999). Efficient Implementation of the Plan Graph in STAN. *Journal of Artificial Intelligence Research*, 10: 87-115.
- Long, D., and Fox, M. (2002). Bridging the Modelling Gap: Examining the Expressiveness of Planning Domain Description Languages. *AI Planning and Scheduling for Autonomy in Space Applications*.
- Long, D. and Fox, M. (2003). The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research*, 20: 1-59.
- Møller, J. and Lichtenberg, J. (1998). Difference decision diagrams. Master's Thesis. Department of Information Technology, Technical University of Denmark.
- Morris, P., Muscettola, N., and Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In *Proc. IJCAI-01*, 494-502.
- Muscettola, N. (1994). HSTS: integrating planning and scheduling. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*, 169-212. Morgan Kaufmann.
- Nguyen, X., and Kambhampati, S. (2001). Reviving Partial Order Planning. In *Proc. IJCAI-01*, 459-466.
- Oddi, A. and Cesta, A. (2000). Incremental forward checking for the disjunctive temporal constraint problem. In *Proc. 14<sup>th</sup> Eur. Conf. on AI*, 108-112.
- Prosser, P. (1993). Hybrid algorithms for constraint satisfaction problems. *Computational Intelligence*, 9(3): 268-299.
- Schiex, T., and Verfaillie, G. (1994). Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. *International Journal of Artificial Intelligence Tools*, 3(2): 187-207.
- Smith, D.E. (2003). The Case for Durative Actions: A Commentary on PDDL2.1. *Journal of Artificial Intelligence Research*, 20: 149-154.
- Smith, D.E., Frank, J., and Jónsson, A.K. (2000). Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review*, 15(1): 61-94.
- Smith, D.E., and Weld, D.S. (1999). Temporal Planning with Mutual Exclusion Reasoning. In *Proc. IJCAI-99*, 326-333.
- Stergiou, K., and Koubarakis, M. (1998). Backtracking Algorithms for Disjunctions of Temporal Constraints. *15<sup>th</sup> National Conference of Artificial Intelligence (AAAI-98)*, 81-117.
- Tsamardinos, I., and Pollack, M. (2003). Efficient Solution Techniques for Disjunctive Temporal Reasoning Problems. *Artificial Intelligence*, 151(1-2): 43-90.