Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming[*]

Vincent Vidal
CRIL - Université d'Artois
rue de l'université - SP16
62307 Lens Cedex, FRANCE
vidal@cril.univ-artois.fr

Héctor Geffner
ICREA & Universitat Pompeu Fabra
Paseo de Circunvalacion 8
08003 Barcelona, SPAIN
hector.geffner@upf.edu

Abstract

A key feature of modern optimal planners such as Graphplan and Blackbox is their ability to prune large parts of the search space. Previous Partial Order Causal Link (POCL) planners provide an alternative branching scheme but lacking comparable pruning mechanisms do not perform as well. In this paper, a domain-independent formulation of temporal planning based on Constraint Programming is introduced that successfully combines a POCL branching scheme with powerful and sound pruning rules. The key novelty in the formulation is the ability to reason about supports, precedences, and causal links involving actions that are not in the plan. Experiments over a wide range of benchmarks show that the resulting optimal temporal planner is much faster than current ones and is competitive with the best parallel planners in the special case in which actions have all the same duration.

Introduction

The search for optimal plans, like the search for optimal solutions in many intractable combinatorial optimization problems, can be understood along two dimensions: the *branching scheme* used for expanding partial solutions, and the *pruning scheme* used for discarding them. Most AI planning frameworks can be understood in these terms. Optimal state-based planners, for example, branch by performing state regression or progression, and prune by comparing the estimated cost of the partial plans with a given bound (Haslum & Geffner 2000). Optimal SAT and CSP planners, on the other hand, branch by picking a variable and trying each of its values, pruning branches and domain values that lead to an inconsistency (Kautz & Selman 1999; Do & Kambhampati 2000). Pruning is a key operation in both cases: in the first, it is the result of the use of explicit lower bounds, in the second, of constraint propagation mechanisms and bounds encoded in the planning graph (Blum & Furst 1995). This pruning power distinguishes modern planners such as Graphplan from its predecessors (whether optimal or not). Indeed the main limitation of traditional Partial Order Causal Link (POCL) planners is that they provide an alternative branching scheme but no comparable pruning mechanisms. The result is that dead-ends are discovered late and the size of the search tree explodes much sooner.

Due to its expressive power, however, POCL planning remains an appealing framework for planning, and in particular temporal planning (Smith, Frank, & Jonsson 2000). The challenge is to close the performance gap that separates POCL planners from modern planners while retaining the optimality guarantees. In this paper, we undertake this challenge, extending a POCL temporal planner with powerful and sound pruning mechanisms based on a constraint programming formulation that integrates existing lower bounds with propagation rules that reason with supports, precedences, and causal links in novel ways. The experiments show that the resulting planner is faster than current optimal temporal planners and is competitive with current parallel planners in the special case in which action durations are all uniform.

A couple of remarks before proceeding. First, the formulation and implementation that we present is not completely general: we deal only with plans in which every (ground) action is executed at most once; plans that we call *canonical* (Geffner 2001). Canonical plans are thus halfway between full temporal planning and scheduling: while in scheduling typically every action is done *exactly once,* and in canonical planning it is done *at most once,* in full temporal planning, it may be done an arbitrary number of times. As we will see, for most benchmarks in sequential, parallel, or temporal planning, the optimal plans are canonical in this sense but they are not always so. Later on we will discuss briefly how to remove this assumption and preliminary results we have obtained in that direction.

The integration of heuristic functions in a POCL planning framework has been pursued recently in (Nguyen & Kambhampati 2001; Younes & Simmons 2003). However, no attempt at the generation of optimal plans is made in these proposals. Here we make use of some of the ideas in (Nguyen & Kambhampati 2001) like the use of structural mutexes for extending the notion of threats in POCL planning, and the use of disjunctive constraints for expressing the possible resolution of threats. Temporal POCL planners featuring constraint propagation mechanisms include IxTET (Laborie & Ghallab 1995) and RAX (Jonsson *et al.* 2000). These

[*]H. Geffner partially supported by Grant TIC2002-04470-C03-02, MCyT, Spain.

planners are more expressive than ours (e.g., in the use of resources), but their pruning mechanisms are weaker as they tend to reason about actions in the current partial plan only. Something similar occurs with formulations of POCL planning as Dynamic CSPs (Joslin & Pollack 1996). A previous CP approach to planning over various *specific* domains is given in (Van Beek & Chen 1999). We borrow some elements from this formulation, like the use of *distances* of various sorts, yet our approach is domain-independent. The broad ideas on which the current proposal is based have been outlined first in (Geffner 2001), and a preliminary implementation for parallel planning was reported earlier in (Palacios & Geffner 2002).

Preview

In order to illustrate the capabilities of the proposed planner, let us consider the problem TOWER-n where the task is to build a tower with n blocks b_1, …, b_n in that order, b_1 on top, which initially lie all on the table. The single optimal plan for this problem involves picking each block b_i from the table and stacking it on block b_{i+1}, in order, from $i = n - 1$ down to 1. The reasoning mechanisms underlying the proposed planner, that we call CPT, yield a solution to this problem *by pure inference and no search.* This is remarkable as the inferences are not trivial and existing optimal planners do not scale up well in this domain (see Table 1). How does CPT do it? First, it infers that each subgoal $on(b_i, b_{i+1})$ must be achieved by the action $stack(b_i, b_{i+1})$ and then that these actions must be ordered sequentially, $stack(b_{n-1}, b_n)$ first, then $stack(b_{n-2}, b_{n-1})$, and so on. CPT also infers that the first action cannot start earlier than $t = 1$, the second not earlier than $t = 3$, etc. Then, after setting the bound $B = 2(n - 1)$ on the makespan to the earliest starting time of the action End, it infers that the starting times of all actions must be set to their lowest possible values, and adds the actions $pick(b_i)$ at their correct times as a result of further reasoning that prunes the other possible supports and times.

Problem	CPU time (sec.)				Makespan
	CPT	BBOX	IPP	TP4	
tower-8	0.33	2.95	0.05	17.68	14
tower-9	0.64	7.28	0.11	887.7	16
tower-10	1.01	13.6	0.38	-	18
tower-11	1.69	28.2	2.26	-	20
tower-12	3.61	-	15.35	-	22
tower-13	5.83	-	123.78	-	24
tower-14	9.70	-	-	-	26
tower-15	13.65	-	-	-	28

Table 1: Results for TOWER-n domain

Table 1 shows results for CPT in relation to other three modern planners: two optimal parallel planners, Blackbox (with Chaff) (Kautz & Selman 1999) and IPP (Koehler *et al.* 1997), and an optimal temporal planner TP4 (Haslum & Geffner 2001). While most domains are not like TOWER-n and require search, the domain illustrates the strength of CPT inference mechanisms that often manage to prune the search space considerably. Over the next few sections we will see how this is achieved and how cost-effective these mechanisms are in other parallel and temporal domains.

Background

The proposed scheme combines lower bounds, a branching scheme that parallels the one used in POCL planning, and a constraint-directed branch-and-bound search.

Lower Bounds

A recent key development in AI planning is the use of *heuristic estimators* automatically extracted from problem encodings (McDermott 1996; Bonet, Loerincs, & Geffner 1997). A parameterized family of tractable admissible heuristics or lower bounds h^m, $m = 1, 2, \ldots$, for sequential and parallel planning is formulated in (Haslum & Geffner 2000). The heuristics h^m recursively approximate *the cost of a set of atoms C by the cost of the most costly subset of size m in C.* For $m = 2$, in the parallel setting, h^m is equivalent to the heuristic encoded in the planning graph. The h^m heuristics are extended in (Haslum & Geffner 2001) to estimate *makespan* (completion time) in a temporal setting. The measures $h^m_T(C)$ are lower bounds on the time needed to make C true from the initial situation. In CPT these heuristics are used in various ways.

Branching

Branching in AI planning is most often discussed in terms of the *space* in which the search for plans is done, with state or directional planners searching in the space of *states*, and partial order planners searching in the space of *plans* (Kambhampati, Knoblock, & Yang 1995). All planners, however, can also be seen as searching in the space of plans, with directional planners taking advantage of a *decomposition property* by which a partial plan tail or head can be summarized by the state obtained by regressing the goal or progressing the initial state. This decomposition is not possible in non-directional partial plans as arising from POCL, SAT, or CSP formulations. In all cases, however, in order to search effectively for optimal plans it is necessary to detect and prune early partial plans that can only lead to solutions with cost exceeding a certain bound B. In state-based planners this is accomplished by comparing the bound B with the value of an explicit evaluation function; in SAT and CSP formulations, this is achieved by means of clauses and constraints. Planning schemes based on POCL branching lack comparable pruning mechanisms and do not perform as well. Recent proposals like (Nguyen & Kambhampati 2001; Younes & Simmons 2003) extend POCL planning with guiding non-admissible heuristics but leave optimality considerations aside. Here we aim to achieve both good performance and optimality in the more general setting of temporal planning.

Temporal Planning

A Strips temporal planning problem is a tuple $P = \langle A, I, O, G \rangle$ where A is a set of ground atoms, $I \subseteq A$ and $G \subseteq A$ represent the initial and goal situations, and O is the set of ground Strips operators, each with precondition, add, and delete list $pre(a)$, $add(a)$, and $del(a)$, and *duration* $dur(a)$. As in Graphplan, two actions a and a' interfere when one deletes a precondition or positive effect of the other. We follow the simple model of time in (Smith & Weld 1999), and define a valid plan as a plan where interfering actions do not overlap in time. We are interested in computing valid plans with minimum *makespan.* When all actions have uniform durations, the model reduces to the standard model of parallel planning.

Temporal POCL Planning

Branching in POCL planning proceeds by picking a 'flaw' (open preconditions or threats) and trying each of the possible *repairs* (Weld 1994; Kambhampati, Knoblock, & Yang 1995). A state or partial plan in the resulting search space corresponds to a set of commitments represented by a tuple $\sigma = \langle Steps, Ord, CL, Open \rangle$, where $Steps$ is the set of actions in the partial plan, Ord is a set of precedence constraints on $Steps$, CL is a set of causal links, and $Open$ is a set of open preconditions (as in most current planners, we assume that all actions are grounded). A state is terminal if it is inconsistent (i.e., the ordering Ord is inconsistent or contains flaws that cannot be fixed) or is a *goal* (is consistent and contains no flaws).

The adaptation of POCL branching to the temporal setting is rather direct (e.g., (Laborie & Ghallab 1995)). Here we consider a simple extension obtained by adding temporal variables $T(a)$ standing for the starting time of action a for $a \in Steps$. These temporal variables have initial domains $T(Start) = 0$, $T(End) = B$, and $T(a) :: [0, B - dur(a)]$ where B is the bound on the makespan ($Start$ and End are the two 'dummy' actions used in POCL planning). The resulting states have the form $\sigma = \langle Steps, Ord_T, CL, Open, T(\cdot) \rangle$ where the qualitative precedence ordering Ord is replaced by the temporal variables $T(a)$, $a \in Steps$, and the set Ord_T of temporal constraints (the precedence ordering Ord from classical POCL planning can be retained but is not strictly necessary).

As before, branching proceeds by picking a 'flaw' in a non-terminal state σ and applying the possible repairs. *Open precondition flaws* $[p]a$ in σ are solved by selecting an action a' that supports p, and adding the causal link $a'[p]a$ to CL and the temporal constraint $T(a') + dur(a') \leq T(a)$ to Ord_T. The action a' is added to $Steps$ if $a' \notin Steps$ and in such case a variable $T(a')$ for a' is created. Similarly, causal link *threats*, i.e., situations in which an action $a \in Steps$ may delete a condition $p \in del(a)$ in a causal link $a_1[p]a_2$ in CL, are solved by adding one of the temporal constraints $T(a) + dur(a) \leq T(a_1)$ or $T(a_2) + dur(a_2) \leq T(a)$ to Ord_T. A terminal state in the resulting space is either a state with an inconsistent set of temporal constraints (a *dead-end*) or a state with a consistent set of temporal constraints and no flaws (a *goal*).

The temporal constraints in Ord_T form a Simple Temporal Problem (STP) (Dechter, Meiri, & Pearl 1991) whose consistency can be tested efficiently by applying a form of constraint propagation known as *bounds consistency* (Lhomme 1993), where the lower and upper bounds $T_{min}(a)$ and $T_{max}(a)$ of the variables $T(a)$ in constraints of the form $T(a) + dur(a) \leq T(a')$ are updated as $T_{max}(a) := \min[T_{max}(a), T_{max}(a') - dur(a)]$ and $T_{min}(a') := \max[T_{min}(a'), T_{min}(a) + dur(a)]$ until a fixed point is reached or a variable domain becomes empty.

With two additional provisions, it is possible to verify that the resulting branching scheme is *sound* and *complete*; i.e., terminal goal-states encode valid temporal plans P with makespan B where actions execute at their earliest possible times, and one such terminal goal state is generated when one such plan exists.

The two required provisions are the following. First, in the absence of a qualitative precedence ordering on actions as in POCL planning, we need to regard an action a deleting the condition p in a causal link $a_1[p]a_2$ as a *threat* when neither of the two temporal conditions $T_{min}(a) + dur(a) \leq T_{min}(a_1)$ and $T_{min}(a_2) + dur(a_2) \leq T_{min}(a)$ hold. This is because the lower bounds T_{min} provide a consistent solution to a STP if the STP is consistent. Second, in accordance with the semantics, we need to ensure that interfering actions do not overlap in time. For that, let us say that a pair of interfering actions are *precondition-interfering* when one action deletes a precondition of the other, and are *effect-interfering* otherwise. It is then sufficient to branch also on a second class of threats; *mutex threats:* pairs of effect-interfering actions a and a' such that neither $T_{min}(a) + dur(a) \leq T_{min}(a')$ nor $T_{min}(a') + dur(a') \leq T_{min}(a)$ hold in σ. Such flaws are solved by adding to Ord_T one of the temporal constraints $T(a) + dur(a) \leq T(a')$ or $T(a') + dur(a') \leq T(a)$.

Modern Constraint-Based Interval (CBI) planners (Jonsson *et al.* 2000; Smith, Frank, & Jonsson 2000) are based on similar ideas and are able to deal with more expressive languages. Yet, as in standard POCL and Dynamic CSP planners (Joslin & Pollack 1996), the following *performance problem* remains: pruning partial plans whose STP network is not consistent does not suffice to match the performance of modern planners. For this, *more powerful representations and inference methods for predicting that all STP networks in the way to the goal will become inconsistent* are needed.

A Constraint Programming Formulation

The performance limitation of current constraint-based POCL planners arises mainly from their limitation *to reason about the actions in the current plan only.* Most often, nothing is inferred about an action a until the action is considered for inclusion in the plan. Still, as we have seen in Section 2, a lot can be inferred about such actions including restrictions about their possible starting times and supporters. Some of this information can actually be inferred before any commitments are made; the lower bounds on the starting times of *all* actions as computed in Graphplan being one example. Yet this is not enough; if similar performance and optimality guarantees are to be achieved in the POCL setting, inferences that take advantage of the commitments made are also necessary. In order to perform such inferences, the representation of the space of possible commitments is crucial. We thus make two changes in relation to the temporal POCL planner above. First, we introduce and reason with variables that involve *all* the actions a in the domain; not only those present in the current plan. And second, for all such actions we introduce variables $S(p,a)$ and $T(p,a)$ that stand for the possibly undetermined action supporting precondition p of a and the possibly undetermined starting time of such an action, and perform limited but useful forms of reasoning over such variables. A causal link $a'[p]a$ thus becomes a constraint $S(p,a) = a'$, which in turn implies that the supporter a' of precondition p of a starts at time $T(p,a) = T(a')$.[1]

An important assumption that we make is that *no (ground) action a in the domain occurs more than once in the plan.* This *canonicity assumption* allows us to collapse the notions of action and action occurrence, leading to a number of simplifications. At the same time, it is a meaningful extension of the more restrictive assumption often found in scheduling research where every action in the domain must occur exactly *once.* From a practical point of view, most current benchmarks, whether temporal or not, admit optimal solutions of this kind, as we will see below. Later on we will discuss ways for relaxing this restriction.

The basic CP formulation of the CPT planner is given in four parts: *preprocessing, variables, constraints, and branching.* After the preprocessing, the variables are created and the constraints are asserted and propagated. If an inconsistency is found, no valid plan for the problem exists. Otherwise, the constraint $T(End) = B$ for the bound B set to the earliest possible starting time of the action End (i.e.; $B = T_{min}(End)$) is asserted and propagated. The branching scheme then takes over and if no solution is found, the process restarts by retracting the constraint $T(End) = B$ and replacing it with $T(End) = B + 1$, and so on.

Preprocessing

In the preprocessing phase, the planner computes the heuristic values $h^2_T(a)$ and $h^2_T(\{p,q\})$ for each action $a \in O$ and each atom pair $\{p,q\}$ as in (Haslum & Geffner 2001). The values provide lower bounds on the times to achieve the preconditions of a and the pair of atoms p,q, from the initial situation I. In addition, we identify the *(structural) mutexes* as the pairs of atoms p,q for which $h^2_T(\{p,q\}) = \infty$. We then say that an action a *e-deletes* an atom p when either a deletes p, a adds an atom q such that q and p are mutex, or a precondition r of a is mutex with p and a does not add p (in all cases p is false after doing a; see (Nguyen & Kambhampati 2001)).

In addition, the simpler heuristic h^1_T is used for defining *distances* between actions (Van Beek & Chen 1999) as follows. For each action $a \in O$, we compute the h^1_T heuristic from an initial situation I_a that includes all facts *except those that are e-deleted by a.* We then set the distances $dist(a,a')$ to the resulting $h^1_T(a')$ values. Clearly, these distances encode lower bounds on the *slack* that can be inserted between the completion of a and the start of a' in any legal plan in which a' follows a.

The distances $dist(a, End)$ and $dist(a, Start)$ are defined in a slightly different way. The former are obtained by running a shortest-path algorithm over a 'relevance graph' where the nodes are the actions $a \in O$ and the action End is the source node. An edge $a \rightarrow a'$ in this graph means that a' is 'relevant' to a (namely that it adds a precondition p of a) and its cost is given by $\delta(a',a) = dur(a') + dist(a',a)$. The distances $dist(a, End)$ are then set to the cost of the shortest-path connecting End to a in this graph. The distances $dist(Start, a)$ are set to $h^2_T(a)$.

Variables and Domains

The state σ of the planner is given by a collection of variables, domains, and constraints. As emphasized above, the variables are defined for each action $a \in O$ and not only for the actions in the current plan. Moreover, variables are created for each precondition p of each action a as indicated below. The domain of variable X is indicated by $D[X]$ or simply as $X :: [X_{min}, X_{max}]$ if X is a numerical variable. The variables, their initial domains, and their meanings are:

- $T(a) :: [0, \infty]$ encodes the starting time of each action a, with $T(Start) = 0$.
- $S(p,a)$ encodes the support of precondition p of action a with initial domain $D[S(p,a)] = O(p)$ where $O(p)$ is the set of actions in O that add p
- $T(p,a) :: [0, \infty]$ encodes the starting time of $S(p,a)$
- $InPlan(a) :: [0,1]$ indicates the presence of a in the plan; $InPlan(Start) = InPlan(End) = 1$ (true)

In addition, the set of actions in the current plan is kept in the variable *Steps*; i.e., $Steps = \{a \mid InPlan(a) = 1\}$. Variables $T(a)$, $S(p,a)$, and $T(p,a)$ associated with actions a which are not yet in the plan are *conditional* in the following sense: these variables and their domains are meaningful only under the assumption that they will be part of the plan. In order to ensure this interpretation, some care needs to be taken in the propagation of constraints as explained below.

Constraints

The constraints correspond basically to disjunctions, rules, and temporal constraints, or their combination. Disjunctions are interpreted constructively: when one disjunct is false, the other is enforced. Similarly for rules: when the antecedent constraint holds, the consequent is enforced. The conditions under which a constraint is regarded as (necessarily) true or false in a state are determined by the nature of the constraint and the domains of the variables; roughly, a constraint is true (false) if it is true (false) for any possible assignment given the domains. E.g., $T(a) < T(a')$ is true if the variable domains are such that $T_{max}(a) < T_{min}(a')$ holds, is false if $T_{min}(a) \geq T_{max}(a')$ holds, and otherwise is *undetermined.*[2] Temporal constraints are propagated by

[1]Propositional 'causal' encodings of Strips planning problems have been formulated and analyzed in (Kautz, McAllester, & Selman 1996; Mali & Kambhampati 1999). Our encodings share a number of features with these formulations but are more compact due to the use of a temporal representation.

[2]Similarly, $T(a) = T(a')$ is true if $T_{min}(a) = T_{max}(a) = T_{min}(a') = T_{max}(a')$ holds, and is false if either $T(a) < T(a')$ or $T(a) > T(a')$ holds. The conditions for enumerated variables like $S(p,a)$ are similar; $S(p,a) = a'$ is true if $D[S(p,a)] = \{a'\}$

bounds consistency as indicated above. In constraints involving terms of the form $op_{a' \in D[S(p,a)]}$, information propagates *from* $S(p,a)$ but not *into* $S(p,a)$; propagation into such variables is achieved by explicit rules with variables $S(p,a)$ on the right hand side. The constraints apply to all actions $a \in O$ and all $p \in pre(a)$; we use $\delta(a,a')$ to stand for $dur(a) + dist(a,a')$.

- **Bounds:** For all $a \in O$, $T(Start) + dist(Start, a) \leq T(a)$ and $T(a) + dist(a, End) \leq T(End)$.
- **Preconditions:** Supporter a' of precondition p of a must precede a by an amount that depends on $\delta(a', a)$:
$$T(a) \geq \min_{a' \in [D(S(p,a)]} (T(a') + \delta(a', a))$$
- **Causal Link Constraints:** for all $a \in O$, $p \in pre(a)$ and a' that e-deletes p, a' precedes $S(p,a)$ or follows a
$$T(a') + dur(a') + \min_{a'' \in D[S(p,a)]} dist(a', a'') \leq T(p,a)$$
$$\vee \; T(a) + dur(a) + dist(a, a') \leq T(a')$$
- **Mutex Constraints:** For effect-interfering a and a'
$$T(a) + \delta(a,a') \leq T(a') \;\vee\; T(a') + \delta(a', a) \leq T(a)$$
- **Support Constraints:** $T(p,a)$ and $S(p,a)$ related by
$$S(p,a) = a' \rightarrow T(p,a) = T(a')$$
$$\min_{a' \in D[S(p,a)]} T(a') \leq T(p,a) \leq \max_{a' \in D[S(p,a)]} T(a')$$
$$T(p,a) \neq T(a') \rightarrow S(p,a) \neq a'$$
$$T(a') + \delta(a', a) > T(a) \rightarrow S(p,a) \neq a'$$

The constraints involving the variables $S(p,a)$ and $T(p,a)$ are *lifted* in the sense that they apply to all possible supporters a' of precondition p of a. As mentioned above, the variables $T(a)$, $T(p,a)$, and $S(p,a)$ are *conditional* when $InPlan(a) = 1$ is neither true or false. They become *in-plan* variables when $InPlan(a) = 1$ becomes true, and *out-plan* variables when $InPlan(a) = 1$ becomes false. Constraints involving in-plan variables only are propagated as usual, and furthermore, an empty domain raises an inconsistency. Constraints involving an out-plan variable, on the other hand, are not propagated. Finally, and most importantly, constraints involving conditional variables associated with the *same action* a and hence the same assumption (namely that a will be part of the plan) are propagated but *only in the direction of the conditional variables.* This ensures that the domain of a conditional variable depends only on the assumption that that particular variable is in the plan and on no other assumption. As a result, *if the domain of a conditional variable associated with an action a becomes empty, it is inferred that the action a cannot be part of the current plan and not that the current partial plan is inconsistent.* More precisely, $InPlan(a)$ is set to 0 if the domain of a conditional variable associated with a becomes empty, and in such case, the action a is removed from the domain of all support variables $S(p,a')$ such that a adds p. On the other hand, when $S(p,a') = a$ holds for some action a' in the plan, $InPlan(a)$ is automatically set to 1. Conditional variables of this type in constraint programming have been considered in (Focacci & Milano 2001).

and is false if $a' \notin D[S(p,a)]$. In all cases, the constraint $\neg C$ is true (false) if C is false (true). In CP, it is common to say that a constraint is *entailed* in a state rather than true (Van Hentenryck, Simonis, & Dincbas 1992). We also note that $T(a) < T(a')$ is true in our modified CP engine when $a' = End$, regardless of the domain of $T(a)$.

Branching

The definition of 'flaws' parallels the one considered above for temporal POCL planning:

- **Support Threats:** a' *threats a support* $S(p,a)$ when both actions a and a' are in the current plan, a' e-deletes p, and neither $T_{min}(a') + dur(a') \leq T_{min}(p,a)$ nor $T_{min}(a) + dur(a) \leq T_{min}(a')$ hold.
- **Open Conditions:** $S(p,a)$ is an *open condition* when $|D[S(p,a)]| > 1$ holds for an action a in the plan.
- **Mutex Threats:** a and a' constitute a *mutex threat* when both actions are in the plan, they are effect-interfering, and neither $T_{min}(a) + dur(a) \leq T_{min}(a')$ nor $T_{min}(a') + dur(a') \leq T_{min}(a)$ hold (two actions are effect-interfering in CPT when one deletes a positive effect of the other, and neither one *e-deletes* a precondition of the other).

Upon selecting a flaw in a state σ, a *binary split* is created which we denote as $[C_1; C_2]$ where C_1 and C_2 are constraints. The first child σ_1 of σ is obtained by adding C_1 to σ and closing the result under the propagation rules; the second child σ_2 of σ is generated by adding the constraint C_2 instead, when the search beneath σ_1 fails. The binary splits generated for each type of flaw are as follows:

- A **Support Threat** $\langle a', S(p,a) \rangle$ generates the split $[T(a') + dur(a') + \min_{a'' \in D[S(p,a)]} dist(a', a'') \leq T(p,a); T(a) + \delta(a,a') \leq T(a')]$
- An **Open Condition** $S(p,a)$ generates the split $[S(p,a) = a'; S(p,a) \neq a']$ for a selected a'
- A **Mutex Threat** $\langle a, a' \rangle$ generates the split $[T(a) + \delta(a,a') \leq T(a'); T(a') + \delta(a',a) \leq T(a)]$

The branching scheme is sound and complete. Soundness follows from the validity of the plan P obtained from a consistent state σ with no flaws by scheduling the in-plan actions a_i at the earliest possible times $t_i = T_{min}(a_i)$. Completeness in turn follows from the soundness of the propagation rules and the validity of the disjunctions $C_1 \vee C_2$ associated with the binary splits $[C_1; C_2]$.

Branching heuristics In each step, the selected flaw is a Support Threat (ST) if one exist, else an Open Condition (OC) if one exists, else a Mutex Threat (MT). The heuristic for selecting among the existing flaws is the following:

- **Support Threats** $\langle a', S(p,a) \rangle$ with minimum slack $\max[slack(a' \prec S(p,a)), slack(a \prec a')]$ selected first (Smith & Cheng 1993), where $slack(a \prec a')$ is $T_{max}(a') - (T_{min}(a) + \delta(a,a'))$ and $slack(a' \prec S(p,a))$ is $T_{max}(p,a) - (T_{min}(a') + \min_{a' \in [D(S(p,a)]} \delta(a', a))$

- **Open conditions** $S(p,a)$ selected latest first; i.e. maximizing the expression $\min_{a' \in D[S(p,a)]} T_{min}(a')$, splitting on the 'arg min' action a' (i.e., creating the split $[S(p,a) = a'; S(p,a) \neq a']$.
- **Mutex Threats** $\langle a, a' \rangle$ selected as they are encountered

The heuristics for STs and OCs have a significant influence on performance but not so the heuristic for MTs (most often no MTs are left after removal of STs and OCs).

Mutex Sets

The code incorporates an enhancement that helps in some domains without representing a significant overhead in others. It has to do with the idea of *mutex sets:* sets M of actions *in the plan,* (not necessarily pairs) such that any two actions in M are interfering. Since such actions cannot overlap, the time window associated with the set of actions M, $\max_{a \in M}(T_{max}(a)+dur(a)) - \min_{a \in M} T_{min}(a)$, must provide enough 'room' for scheduling all actions in $a \in M$ in sequence. Taking into account the pre-computed distances, a lower bound $\Delta(M)$ for the time needed for scheduling all actions in M is given by

$$\sum_{a \in M} [dur(a) + \min_{a' \in M | a' \neq a} dist(a,a')] - \max_{\{a,a'\} \subseteq M} dist(a,a')$$

which expresses a lower bound on the time needed to schedule all the actions in M, one before another, except for the action scheduled last. With these lower bounds, we define the *Mutex Set* constraint as

$$\max_{a' \in M} T(a') - \min_{a'' \in M} T(a'') \geq \Delta(M)$$

and apply it to *some* mutex sets M identified from the actions $Steps$ in the plan in a greedy fashion, as described below (computing the largest mutex sets in the plan seems too expensive). The idea of mutex sets is adapted from similar concepts used in constraint-based scheduling such as *edge-finding;* see (Carlier & Pinson 1989; Baptiste, Le Pape, & Nuijten 2001; Laborie 2003).

- **Global Mutex Sets** M_i are built greedily as new actions are added to $Steps$. Initially a single mutex set M_0 with the $Start$ and End actions is defined; then any time an action a is added to $Steps$, a is added to each existing mutex set M_i, $i = 0, \ldots, k$ such that a is interfering with each action a' in M_i, and a new mutex set M_{k+1} is created with a only when a cannot be added to any existing mutex set. The mutex set constraint is enforced for each such set M_i.
- **Causal Link Mutex Sets** M^- and M^+ are defined also for each 'causal link' $S(p,a)[p]a$ in the plan. Initially, these sets are empty, then when a new action a' is added to the plan that e-deletes p and cannot follow a (resp. cannot precede $S(p,a)$), a is added to M^- (resp. to M^+) if a is interfering with each action in M^- (resp. in M^+). For these mutex sets M^+ and M^-, the following *CL Mutex Set constraint* is enforced, which unlike the mutex set constraint above, not only detects inconsistencies, but also prunes the bounds of the temporal variables $T(p,a)$ and $T(a)$:

$$\min_{a' \in M^-} T(a') + \Delta(M^-) \leq T(p,a) \ \wedge$$

$$T(a) + dur(a) \leq \max_{a' \in M^+} [T(a') + dur(a')] - \Delta(M^+)$$

In addition, for all a' in the plan that e-delete p that can follow $S(p,a)$ and precede a, we evaluate the consistency of the mutex set $M^- \cup \{a'\}$ (resp. $M^+ \cup \{a'\}$) if a' is interfering with each action in M^- (resp. M^+). If the set is inconsistent (i.e., it violates the mutex constraint), then it is inferred that a' must follow a (resp. must precede $S(p,a)$).

Implementation

The CPT planner has been implemented using the Choco CP library (Laburthe 2000) that operates on top of the Claire programming language (Caseau, Josset, & Laburthe 1999) that compiles into C++. In early stages of the implementation, we wrote the constraints in Choco in a way that resembled the formulation above, yet we progressively moved to an implementation based on propagation rules that avoids unnecessary checks and triggerings, and speeds up the propagations. The current implementation is a collection of rules which are triggered by the event mechanism of Choco. Updates on lower bounds, upper bounds, and domain values are recorded in event queues, where similar events are 'collapsed'; e.g., if the lower bound of a variable X is increased successively from 1 to 2, and then from 2 to 3 before the first event is dequeued, only one event is stored, stating that the lower bound of X is increased from 1 to 3. When an event is dequeued, the relevant rules are triggered, performing the appropriate propagations (updates on variables constrained by the modified variables, ...). The only constraints not re-implemented in terms of rules are the dynamic constraints; namely those that are posted as a result of branching. We modified the Choco engine so that these constraints can be backtracked upon inconsistencies, and also for enforcing the semantics of conditional variables. As stated above, for the latter an empty domain does not raise an inconsistency but forces an action out of the plan. On temporal variables, the conditional behavior is obtained by handling those variables ourselves; support conditional variables, on the other hand, are treated as normal CP variables with a dummy action α added to their domains, with $D[S(p,a)] = \{\alpha\}$ meaning that p cannot be supported by any action. The $InPlan(a)$ variables are not implemented as CP variables either; the information about the status of actions in the plan is compiled in the code of the propagation rules. The code incorporates optimizations which we lack the space to discuss here like the use of a larger set of (redundant but effective) 'threats' on which to branch. The code will be made available for download.

Experimental Results

The experiments have been obtained using a Pentium IV machine running at 1.6Ghz, with 512Mb of RAM, under

Linux. The time limit for each problem is one hour. Table 3 compares CPT , Blackbox (with Chaff), IPP, and TP4 over parallel domains, while Table 4 compares CPT and TP4 over temporal domains. These are all optimal planners. The times in all cases include preprocessing. The parallel domains include blocks and logistic instances from the Blackbox distribution. The rest of the instances are from the 3rd Int. Planning Competition (Long & Fox 2003a). The tables show that CPT runtimes are close to Blackbox over the parallel domains, and can even solve problems like bw-large.c and satellite11 that Blackbox cannot solve (the good performance of CPT in blocks owes a lot to the use of mutex sets). CPT is sometimes slower than IPP but can solve many more problems while it clearly dominates TP4 over all parallel and temporal domains, and expands much fewer nodes (these are the numbers in parenthesis). As discussed in (Haslum & Geffner 2001), the problem of state-based temporal planners such as TP4 is their *branching factor* which may be exponential in the number of primitive actions in the domain. In CPT, the branching factor is two, and after every branching decision, a powerful pruning mechanism is applied.

As Table 2 shows, CPT seems to dominate also LPGP, a recent temporal planner that optimizes the number of steps in the plan rather than the makespan (Long & Fox 2003b). We only consider the instances reported for LPGP on a similar machine. The shorter makespans for LPGP in one domain (satellite), arise from slight differences in the semantics (namely, LPGP follows the PDDL2.1 semantics (Fox & Long 2003) where interfering actions may overlap, e.g., when preconditions do not have to be preserved throughout the execution of the action).

Strips	CPU time (sec.)				Makespan
problems	CPT	BBOX	IPP	TP4	
bw.12step	0.21	0.26	0.03	0.08	12
bw.large.a	0.44	1.13	0.07	0.08	12
bw.large.b	1.75	17.94	2.33	-	18
bw.large.c	231.22	-	-	-	28
rocket.a	0.28	0.38	7.97	44.20	7
rocket.b	0.24	0.45	11.95	31.83	7
log.a	0.70	0.47	781.13	-	11
log.b	0.90	0.91	2099.89	-	13
log.c	1.43	1.46	-	-	13
log.d	29.03	3.73	-	-	14
zeno7	0.84	0.67	0.05	1.76	6
zeno8	5.39	1.59	0.22	166.22	5
zeno9	6.41	2.54	0.68	-	6
zeno10	6.84	4.01	221.32	-	6
zeno11	14.90	5.60	31.06	-	6
zeno12	16.39	11.10	-	-	6
zeno13	45.97	11.42	-	-	7
driver7	0.24	0.24	0.15	22.98	6
driver8	0.30	0.40	3.53	33.59	7
driver9	1.46	1.55	11.26	2979.66	10
driver10	1.02	1.00	17.06	1823.16	7
driver11	4.33	2.67	2.26	1259.06	9
satellite3	0.12	0.26	0.03	0.08	6
satellite4	0.40	1.39	7.28	755.08	10
satellite5	0.99	1.50	145.67	-	7
satellite6	0.56	1.34	90.46	-	8
satellite7	1.55	1.80	1039.23	-	6
satellite8	101.18	235.13	-	-	8
satellite9	8.52	4.68	-	-	6
satellite10	185.90	42.35	-	-	8
satellite11	22.51	-	-	-	8

Table 3: Results for parallel domains

Discussion

We have developed a domain-independent optimal POCL temporal planner based on constraint programming that integrates existing lower bounds with novel representations and propagation rules that manage to prune the search space considerably. The experiments show that the resulting planner is faster than current optimal temporal planners and competitive with the best parallel planners. The formulation exploits the canonicity restriction that no (ground) action a in the domain occurs more than once in the plan. This restriction allows us to collapse the notions of action and action occurrence, leading to a number of simplifications. We are currently working on a formulation that removes this restriction. The formulation is based on distinguishing *action types* from *action tokens*. Plans contain only action tokens which are all instances of the fixed set of action types defined by the initial set of operators. Action tokens are created dynamically from action types as in POCL planning; namely, as new action instances that support a selected open condition. In this setting, however, the creation of new instances takes the form of a 'cloning' operation: for the new instance a' of type a, the variables $T(a')$, $S(p, a')$, and $T(p, a')$ are created as fresh copies of the variables $T(a)$, $S(p, a)$, and $T(p, a)$ with their corresponding domains, where p is a precondition of a. In addition, the new token a' is added as an independent action to all support domains that include the action type a. The resulting scheme can actually be seen as providing a lazy implementation of a planning domain with an infinite number of action instances, with the action types summarizing the domains of the action instances that have not been yet used in the plan. For the parallel domains, the performance degradation is moderate (runtimes do not appear to increase in more than 15% over the instances considered in this paper), while for temporal domains, it is greater (although the resulting planner still performs much better than current optimal planners). We are still tuning the non-canonical planner and plan to report the relevant details elsewhere.

References

Baptiste, P.; Le Pape, C.; and Nuijten, W. 2001. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, 1636–1642.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*, 714–719. MIT Press.

Carlier, J., and Pinson, E. 1989. An algorithm for solving the job shop scheduling problem. *Management Science* 35(2).

Caseau, Y.; Josset, F. X.; and Laburthe, F. 1999. Claire: Com-

	zeno4	zeno5	zeno6	driver1	rover1	rover2	rover3	rover4	satellite1	satellite2	satellite3
CPT	4.59 (522)	3.83 (400)	1.78 (323)	0.06 (91)	0.12 (53)	0.07 (43)	0.11 (53)	0.09 (45)	0.05 (46)	0.95 (70)	0.20 (34)
LPGP	65.32 (740)	43.83 (583)	57.61 (350)	0.33 (91)	0.30 (55)	0.24 (44)	0.44 (58)	0.40 (47)	0.17 (41)	24.15 (65)	62.22 (29)

Table 2: Comparison CPT vs. LPGP (CPU time in sec., makespan in parenthesis)

Temporal	CPU time (sec.) (+number of states)		Makespan
problems	CPT	TP4	
zeno1	0.06 (2)	0.05 (4)	173
zeno2	0.95 (892)	1.23 (17124)	592
zeno3	0.50 (4)	0.05 (618)	280
zeno4	4.59 (2233)	-	522
zeno5	3.83 (124)	34.78 (595988)	400
zeno6	1.78 (54)	6.03 (116715)	323
zeno7	77.58 (45187)	-	665
zeno8	265.93 (78044)	-	522
zeno9	1522.24 (432210)	-	522
zeno10	82.62 (12692)	-	453
zeno11	116.15 (874)	-	423
driver1	0.06 (6)	0.05 (49)	91
driver2	734.98 (724327)	458.19 (17444608)	92
driver3	0.12 (11)	0.05 (621)	40
driver4	91.32 (54350)	-	52
driver5	0.40 (152)	-	51
driver6	111.10 (59702)	-	52
driver7	0.59 (103)	20.79 (323963)	40
driver8	-	-	-
driver9	493.91 (137716)	-	92
driver10	8.75 (1517)	-	38
satellite1	0.05 (5)	0.05 (80)	46
satellite2	0.95 (1435)	8.45 (712294)	70
satellite3	0.20 (26)	0.05 (21143)	34
satellite4	4.36 (5257)	-	58
satellite5	2.32 (1191)	-	36
satellite6	0.82 (47)	-	46
satellite7	2.36 (325)	-	34
satellite8	3324.92 (827408)	-	46
satellite9	8.84 (516)	-	34
satellite10	2160.24 (261474)	-	43

Table 4: Results for temporal domains

bining sets, search and rules to better express algorithms. In *Proceedings of the Int. Conf. on Logic Programming*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Do, M. B., and Kambhampati, S. 2000. Solving planning-graph by compiling it into CSP. In *Proc. AIPS-00*, 82–91.

Focacci, F., and Milano, M. 2001. Connections and integrations of dynamic programming and constraint programming. In *Proc. of the Int. Workshop on Integration of AI and OR techniques (CP-AI-OR'01)*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20.

Geffner, H. 2001. Planning as branch and bound and its relation to constraint-based approaches. Technical report, Universidad Simón Bolívar. At www.tecn.upf.es/∼hgeffner.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proc. AIPS-2000*, 70–82.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proc. ECP-01*, 121–132.

Jonsson, A.; Morris, P.; Muscettola, N.; and Rajan, K. 2000. Planning in interplanetary space. In *Proc. AIPS-2000*, 177–186.

Joslin, D., and Pollack, M. E. 1996. Is "early commitment" in plan generation ever a good idea? In *Proc. AAAI-96*, 1188–1193.

Kambhampati, S.; Knoblock, C.; and Yang, Q. 1995. Planning as refinement search. *AI* 76(1-2):167–238.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and Graph-based planning. *Proc. IJCAI-99*, 318–327.

Kautz, H.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proc. KR'96*, 374–384.

Koehler, J.; Nebel, B.; Hoffman, J.; and Dimopoulos, Y. 1997. Extending planning graphs to an ADL subset. In Proc. ECP-97. Lect. Notes in AI 1348, 273–285.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resources constraints. *Proc. IJCAI-95*, 1643–1649.

Laborie, P. 2003. Algorithms for propagating resource constraints in AI planning and scheduling. *AI* 143:151–188.

Laburthe, F. 2000. Choco: implementing a CP kernel. In *Proceedings CP-00, Lecture Notes in CS, Vol 1894*. Springer.

Lhomme, O. 1993. Consistency techniques for numeric CSPs. In *Proceedings IJCAI-93*, 232–238. Morgan Kaufmann.

Long, D., and Fox, M. 2003a. The 3rd international planning competition: Results and analysis. *JAIR* 20:1–59.

Long, D., and Fox, M. 2003b. Exploiting a graphplan framework in temporal planning. In *Proceedings ICAPS 2003*, 52–61.

Mali, A., and Kambhampati, S. 1999. On the utility of plan-space (causal) encodings. In *Proceedings AAAI-99*, 557–563.

McDermott, D. 1996. A heuristic estimator for means-ends analysis in planning. In *Proc. AIPS-96*.

Nguyen, X. L., and Kambhampati, S. 2001. Reviving partial order planning. In *Proc. IJCAI-01*.

Palacios, H., and Geffner, H. 2002. Planning as branch and bound: A constraint programming implementation. In *Proc. XXVIII Conf. Latinoamericana de Informática*, 239–251.

Smith, S., and Cheng, C. 1993. Slack-based heuristics for the constraint satisfaction scheduling. In *Proc. AAAI-93*, 139–144.

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. IJCAI-99*, 326–337.

Smith, D.; Frank, J.; and Jonsson, A. 2000. Bridging the gap between planning and scheduling. *Know. Eng. Review* 15(1).

Van Beek, P., and Chen, X. 1999. CPlan: a constraint programming approach to planning. In *Proc. AAAI-99*, 585–590.

Van Hentenryck, P.; Simonis, H.; and Dincbas, M. 1992. Constraint satisfaction using constraint logic programming. *Artificial Intelligence* 58(1–3):113–159.

Weld, D. S. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27–61.

Younes, B. L. S., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *JAIR* 20:405–430.