Available online at www.sciencedirect.com
SCIENCE DIRECT®
Artificial Intelligence 170 (2006) 298–335
Artificial Intelligence
www.elsevier.com/locate/artint

Branching and pruning: An optimal temporal POCL planner based on constraint programming

Vincent Vidal [a,*], Héctor Geffner [b]

[a] *CRIL – Université d'Artois, rue de l'université – SP16, 62307 Lens Cedex, France*
[b] *ICREA – Universitat Pompeu Fabra, Paseo de Circunvalacion 8, 08003 Barcelona, Spain*

Received 7 December 2004; received in revised form 28 July 2005; accepted 18 August 2005

Available online 29 September 2005

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.[1]

Keywords: Planning; Constraint programming; Temporal reasoning

* Corresponding author.
E-mail addresses: vidal@cril.univ-artois.fr (V. Vidal), hector.geffner@upf.edu (H. Geffner).

[1] This paper extends [V. Vidal, H. Geffner, Branching and pruning: An optimal temporal POCL planner based on constraint programming, in: Proceedings of AAAI-2004, San Jose, CA, 2004, pp. 570–577] by removing the canonicity restriction in the generation of plans. This is a restriction that forces every (ground) action in the domain to be done at most once in the plan. See the text for details.

0004-3702/$ – see front matter
doi:10.1016/j.artint.2005.08.004

1. 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 [16]. 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 [10,23]. 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 [3]. 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 [32,46] 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 [39]. 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.

The proposed scheme shows also the appeal of constraint-programming branch-and-prune formulations for combinatorial optimization problems in which the definition of explicit and informative lower bound functions is difficult to come by [8,12,43]. Indeed, informative admissible heuristics for estimating the completion time of partial POCL plans do not exist, but still we show that suitably chosen constraints and propagation rules may yield an equivalent pruning power.

The integration of heuristic functions in a POCL planning framework has been pursued recently in [35,47]. However, no attempt at the generation of optimal plans is made in these proposals. Here we make use of some of the ideas in [35] 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 [27], ZENO [37] and RAX [18]. These 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: CSPs in which the set of variables and constraints is not determined a priori but gets expanded until a failure is detected or a fixed point is reached [19]. In such cases, the number of potential CSPs to be explored is exponential and for attaining good performance it is

not possible to reason only within the 'current' CSP; it is necessary to reason also over its possible refinements. This is what GRAPHPLAN does when it builds the planning graph: it reasons, in a limited way, about all possible plans, and this is also what is achieved in different ways in our formulation. A previous CP approach to planning over various *specific* domains is given in [42]. 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 [15], and a preliminary implementation for parallel planning was reported earlier in [36]. Here this formulation is extended in a number of ways and a new planner has been implemented over the CHOCO CP library [28] that operates on top of the CLAIRE programming language [7]. This formulation first appeared in [45] along with a restriction on the types of temporal plans that could be generated; namely only *canonical plans* where every ground action in the domain was done *at most* once. This restriction is a slight generalization of the situation most commonly found in scheduling where every action or task has to be done *exactly* once [2,6]. In this paper, this restriction is removed and all empirical results, except where otherwise noted, refer to this general, non-canonical temporal planner called still CPT.

2. Preview

In order to illustrate the capabilities of the proposed planner, we consider the class of planning problems TOWER-n where the task is to build a tower with n blocks $b_1, \ldots, b_n$ in that order, with b_1 on top, all blocks initially 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}, from $i = n-1$ until $i = 1$. 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 quite remarkable as the inferences are not trivial and existing optimal planners do not scale up well over these problems (see Table 1). How does CPT do it? First, it is inferred that each subgoal $on(b_i, b_{i+1})$ must be achieved by the action $stack(b_i, b_{i+1})$. This inference is simple as there is a single possible supporter in each case. More interestingly, it is then inferred that these stack operations must be ordered sequentially in descending order of i; namely, $stack(b_{n-1}, b_n)$ first, then $stack(b_{n-2}, b_{n-1})$, and so on, until $stack(b_1, b_2)$. This is inferred by reasoning with and resolving the threats affecting the causal links $stack(b_i, b_{i+1})[on(b_i, b_{i+1})]End$.[2] Moreover, it is also inferred that the first action in the sequence cannot occur earlier than $t = 1$, the second action not earlier than $t = 3$, the third not earlier than $t = 5$, and so on, and that the *End* action cannot start earlier than $2(n-1)$, the optimal time bound. This is because as part of the preprocessing CPT infers that no stack action can be done before $t = 1$ and that at least a unit of time must separate the ending of one stack action and the beginning of a new one (all actions are assumed to have unit durations in the example).

[2] We use the notation $a[p]a'$ for causal links in which action a supports precondition p of a', often denoted in the literature as $a \xrightarrow{p} a'$.

All these inferences result from the domain constraints and propagation mechanisms before even a search bound B on the allowed makespan of the plan is fixed. After the first bound $B = 2(n-1)$ is chosen (this is the earliest time at which the action *End* can start), further inferences are made. First, the starting times $T(a_k)$ of all the actions a_i in the stack sequence above become fixed to their earliest possible starting times resulting in $T(a_k) = 1 + 2k$, for $k = 1, \ldots, n-1$, where a_k is the kth action in the sequence (namely $a_k = \mathit{stack}(b_{n-k}, b_{n-k+1})$). Then the $\mathit{pickup}(b_{n-1})$, $\mathit{pickup}(b_{n-2})$, ... sequence gets added to the set of actions in the plan at their correct starting times as a result of further reasoning that prunes the other possible supports and times. For example, the precondition $\mathit{clear}(b_n)$ for the first action $a_1 = \mathit{stack}(b_{n-1}, b_n)$ in the sequence can be supported by a number of $\mathit{unstack}(*, b_n)$ and $\mathit{stack}(b_n, *)$ actions, and by *Start*. However, since any such supporter a' must precede a_1 and $T(a_1) = 1$ is already fixed, $T(a') < 1$ must hold, leaving $a' = \mathit{Start}$ as the only possible supporter (at preprocessing, lower bounds on the starting time of actions are computed from which it is known that $T(a') < 1$ is true only for *Start* and *pickup* actions). For similar reasons, all supporters $\mathit{unstack}(b_{n-1}, *)$ for the other precondition $\mathit{holding}(b_{n-1})$ of a_1 are pruned, leaving $a'_1 = \mathit{pickup}(b_{n-1})$ as the only possible support. The process repeats for the preconditions of $a'_1 = \mathit{pickup}(b_{n-1})$ with all supporters a' different than *Start* being pruned as well.

At this point a number of actions and causal links in the plan have been inferred with no commitments made except for the bound B. In particular, due to the causal links going into the actions $\mathit{pickup}(b_{n-1})$ and $\mathit{stack}(b_{n-1}, b_n)$ already fixed at the times $t = 0$ and $t = 1$ respectively, and the fact that all actions a' whether in the plan or not (except for these two and *Start*), threat these causal links but cannot precede both actions, the starting times $T(a')$ of such actions a' are pushed to times $t = 2$ or higher. The result is that the only supporters left for the preconditions $\mathit{clear}(b_{n-1})$ and $\mathit{holding}(b_{n-2})$ of the next stack action in the sequence, $a_2 = \mathit{stack}(b_{n-2}, b_{n-1})$, scheduled at time $t = 3$, end up being the actions $a_1 = \mathit{stack}(b_{n-1}, b_n)$ at $t = 1$ and $\mathit{pickup}(b_{n-2})$ at time $t = 2$. To illustrate this, consider the possible supporters a' of the precondition $\mathit{clear}(b_{n-1})$ of a_2 *different* than a_1 (namely *Start*, $\mathit{unstack}(*, b_{n-1})$, and $\mathit{stack}(b_{n-1}, *)$ actions) and the causal link $a'[\mathit{clear}(b_{n-1})]a_2$. Clearly, for avoiding the action a_1 at time $t = 1$ from threatening this link, one of the precedences $a_1 \prec a'$ or $a_2 \prec a_1$ must hold, but since the latter disjunct is false and $a' \prec a_2$ must hold too, we get $T(a') = 2$ which is not possible for any such supporter a'. The supporter $\mathit{pickup}(b_{n-2})$ for precondition $\mathit{holding}(b_{n-2})$ of a_2 is fixed at time $t = 2$ in a similar way, and the process repeats for all other stack actions in the sequence until all actions have their start times and supporters fixed and no flaw in the plan is left.

Table 1 shows results for CPT in relation to other three modern planners: two optimal parallel planners, BLACKBOX [23] (with CHAFF [34]) and IPP [24], and an optimal temporal planner TP4'04 [17]. 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.

Table 1
Results for TOWER-n domain

	CPU time (sec.)					Makespan
	CPT	BLACKBOX	SATPLAN	IPP	TP4	
tower02	0.00	0.00	0.13	0.00	0.00	2
tower03	0.00	0.00	0.13	0.00	0.00	4
tower04	0.01	0.02	0.16	0.00	0.01	6
tower05	0.01	0.08	0.32	0.00	0.03	8
tower06	0.02	0.24	3.30	0.00	0.08	10
tower07	0.03	0.75	39.75	0.01	0.32	12
tower08	0.06	1.85	236.02	0.01	1.75	14
tower09	0.08	3.56	665.76	0.04	12.11	16
tower10	0.11	7.07	1229.22	0.19	103.63	18
tower11	0.17	13.92	–	1.10	1096.08	20
tower12	0.26	26.93	–	7.42	–	22
tower13	0.36	52.16	–	61.32	–	24
tower14	0.54	99.15	–	535.45	–	26
tower15	0.80	–	–	–	–	28
tower16	1.10	–	–	–	–	30
tower17	1.47	–	–	–	–	32
tower18	1.89	–	–	–	–	34
tower19	2.46	–	–	–	–	36
tower20	3.41	–	–	–	–	38
tower21	4.40	–	–	–	–	40
tower22	5.69	–	–	–	–	42

3. Background

The proposed scheme for optimal temporal planning combines three elements: lower bounds automatically extracted from planning problems, a branching scheme that parallels the one used in POCL planning, and a constraint-directed branch-and-bound search. We review these topics over the next sections.

3.1. Lower bounds

A recent key development in AI planning is the use of *heuristic estimators* automatically extracted from problem encodings [5,33]. A parameterized family of lower bounds or admissible heuristics h^m, $m = 1, 2, \ldots$, for sequential and parallel planning is formulated in [16]. The heuristics $h^m(C)$ recursively approximate *the cost of achieving a set of atoms C from an initial state s_0 by the cost of achieving the most costly subset of size $m' \leqslant m$ in C*. For example, for $m = 1$, the heuristic h^m approximates the cost of achieving a set of atoms by the cost of achieving the most costly *atom* in the set. For both sequential and parallel Strips planning, h^m for $m = 1$ is thus given by the equation

$$h^1(C) = \begin{cases} 0 & \text{if } C \subseteq s_0, \text{ else} \\ \min_{o \in O(p)} [1 + h^1(pre(o))] & \text{if } C = \{p\}, \text{ else} \\ \max_{p \in C} h^1(\{p\}) & \text{if } |C| > 1, \end{cases} \quad (1)$$

where p is an atom and $O(p)$ stands for the operators o that add p (h^1 is also known as the h_{max} heuristic; e.g., [4]). The estimators h^m for sequential and parallel planning are equal for $m = 1$ but become different for higher values of m (recall that cost in the sequential and parallel settings refer to number of actions and number of time steps in the plan respectively). Moreover, for $m = 2$, the parallel estimator h^m is equivalent to the heuristic implicitly computed by GRAPHPLAN in the construction of the planning graph: namely, $h^m(A)$ for a set of atoms A is equivalent to the index of the first propositional layer that contains the atoms in A without a mutex [16].

From a computational point of view, for a fixed m, the heuristics h^m are polynomial in both the number of actions and the number of atoms in the problem, and they can be computed by a shortest-path algorithm over a graph in which the nodes are given by the sets of at most m atoms [16].

The heuristics h^m have also been extended to estimate *makespan* (completion time) in a temporal setting where actions can be executed concurrently and have different durations [17]. The equation for $m = 1$ in that setting becomes

$$h_T^1(C) = \begin{cases} 0 & \text{if } C \subseteq s_0, \text{ else} \\ \min_{o \in O(p)}[dur(o) + h_T^1(pre(o))] & \text{if } C = \{p\}, \text{ else} \\ \max_{p \in C} h_T^1(\{p\}) & \text{if } |C| > 1, \end{cases} \tag{2}$$

where the only change from the *parallel* estimator h^1 is the substitution of the fixed cost 1 by the duration $dur(a)$ of the action a. For $m = 2$, the temporal estimator h_T^2 departs from parallel h^2 in other ways; see [17] for details. The measures $h_T^m(C)$ are lower bounds on the time needed to make C true from the initial situation s_0. In CPT we use the h_T^2 heuristic for initializing the value of certain temporal variables, and enforce a version of the h_T^1 heuristic over partial plans through a set of 'precondition' constraints.

3.2. 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 in the space of plans [20,21]. This perspective has been very useful in planning, although it does not always make explicit what these various approaches to planning have in common, including the more recent SAT and CSP formulations. All planners, indeed, search in the space of plans (solutions); directional planners, however, are able to exploit a *decomposition property* for which a partial plan tail or head σ can be summarized by the state s^σ obtained by regressing the goal or progressing the initial state through σ. 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 partial plans σ 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 $f(\sigma)$ that adds up the accumulated cost $g(\sigma)$ of the plan and an estimate $h(s^\sigma)$ of the 'cost to go'. In SAT and CSP formulations, a constraint $f^*(\sigma) \leqslant B$ or $f^*(\sigma) = B$ defining the feasible partial plans σ is explicitly added (f^* stands for the optimal cost function); e.g., in SAT formula-

tions unit clauses like p_{10} and q_{10} are added when searching for plans leading to the goals p and q with costs not exceeding $B = 10$. Planning schemes based on POCL branching, on the other hand, have lacked comparable pruning mechanisms. Recent proposals like [35,47] extend POCL planning with guiding non-admissible heuristics, leaving optimality considerations aside. Here we aim to achieve both good performance and optimality in the more general setting of temporal planning.

3.3. Temporal planning

We consider a simple extension of the Strips language that accommodates concurrent actions with integer durations. A number of extensions could easily be added but we have chosen to keep the model as simple as possible focusing instead on performance and optimality issues. The appeal of POCL planning for rich temporal settings is discussed in [39]. A temporal planning problem is a tuple $P = \langle A, I, O, G\rangle$ where A is a set of ground atoms (the boolean variables of interest), $I \subseteq A$ 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 is common in POCL Planning, we also consider two dummy actions *Start* and *End* with zero durations, the first with an empty precondition and effect I; the latter with precondition G and empty effects. 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 [40], and define a valid plan as a plan where interfering actions do not overlap in time. In other words, we assume that the preconditions need to hold until the end of the action, and that the effects also hold at the end and cannot be deleted during the execution by a concurrent action. We are interested in computing valid plans with minimum *makespan*. Other models of concurrency could also be used (see [14]). When all actions have uniform durations, the model reduces to the standard model of parallel planning.

A *schedule* P is a finite set of time stamped actions $\langle a_i, t_i\rangle$, $i = 1, \ldots, n$, where a_i is an action and t_i is a non-negative integer indicating the starting time of a_i (its ending time is $t_i + dur(a_i)$). P must include the *Start* and *End* actions, the former with time tag 0. The same action (except for these two) can be executed more than once in P if $a_i = a_j$ for $i \neq j$. In such a case, a_i and a_j refer to two *occurrences* of the same action. Two action occurrences a_i and a_j *overlap* in P if one starts before the other ends; namely if $[t_i, t_i + dur(a_i)] \cap [t_j, t_j + dur(a_j)]$ contains more than one time point.

A schedule P is a *valid plan* iff interfering actions do not overlap in P and for every action occurrence a_i in P its preconditions $p \in pre(a)$ are true at time t_i. This condition is inductively defined as follows: p is true at time $t = 0$ iff $p \in I$, and p is true at time $t > 0$ if either p is true at time $t - 1$ and no action a in P ending at t deletes p, or some action a' in P ending at t adds p.

The *makespan* of a plan P is the time tag of the *End* action. An optimal temporal planner computes valid plans with minimum makespan. For this, it is actually sufficient to have a planner that is sound and complete in the following sense: a valid plan with makespan equal to a given bound B is found iff one such plan exists. There are then many strategies for adjusting the bound B so that an optimal makespan is produced; e.g., the bound may be increased until a plan is found, or can be decreased until no plan is found, etc.

3.4. POCL planning

A partial plan or state σ in classical POCL planning corresponds to a set of commitments represented by a tuple $\sigma = \langle \mathit{Steps}, \mathit{Ord}, \mathit{CL}, \mathit{Open} \rangle$, where *Steps* is the set of actions in the partial plan σ, *Ord* is a set of precedence constraints on *Steps*, *CL* is a set of causal links, and *Open* is a set of open preconditions [20,32,46] (we assume that actions are all grounded). A precedence constraint $a \prec a'$ states that action a precedes action a' in the plan, a *causal link* $a[p]a'$ states that action a supports the precondition p of action a in σ, while an *open precondition* $[p]a$ states that action a in the plan has a precondition p that is not yet supported. The initial state σ_0 is given by the tuple $\langle \{\mathit{Start}, \mathit{End}\}, \{\mathit{Start} \prec \mathit{End}\}, \emptyset, \{[G_1]\mathit{End}, \ldots, [G_m]\mathit{End}\} \rangle$ where $G_1, G_2, \ldots, G_m$ are the top level goals in G.

Branching in POCL planning proceeds by picking a 'flaw' in a non-terminal state σ and applying the possible *repairs* [20,46]. Flaws are of two types. *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 precedence constraint $a' \prec a$ to *Ord* (a' should also be added to *Steps* if $a' \notin \mathit{Steps}$). Similarly, *threats*—which refer to situations in which an action $a \in \mathit{Steps}$ deletes the condition p in a causal link $a_1[p]a_2$ in *CL* with the ordering $a_1 \prec a' \prec a_2$ consistent with *Ord*—are solved by placing one of the precedence constraint $a' \prec a_1$ or $a_2 \prec a'$ in *Ord*. 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).

4. Temporal POCL planning

POCL branching can be adapted to the temporal setting in a direct way (e.g., [27]). While extensions to rich temporal settings have been considered in [18,37,39], here we consider a simple extension obtained by the addition of temporal variables $T(a)$ for each of the actions a in the current state σ (i.e., $a \in \mathit{Steps}$), where $T(a)$ stands for the starting time of a. These temporal variables have initial domains $T(\mathit{Start}) = 0$, $T(\mathit{End}) = B$, and $T(a) :: [0, B - \mathit{dur}(a)]$ where B is the bound on the makespan (*Start* and *End* are the two 'dummy' actions used in POCL planning). The resulting states σ have the form $\sigma = \langle \mathit{Steps}, \mathit{Ord}_T, \mathit{CL}, \mathit{Open}, T(\cdot) \rangle$ where the qualitative precedence ordering *Ord* has been replaced by the set of temporal variables $T(a)$, $a \in \mathit{Steps}$ and their domains, along with a set Ord_T of temporal constraints over them. A precedence constraint stating that action a precedes action a' becomes the temporal constraint $T(a) + \mathit{dur}(a) \leqslant T(a')$. The qualitative precedence relation *Ord* from classical POCL planning can be preserved although this is not strictly necessary. Initially, the set Ord_T is empty.

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 $T(a') + \mathit{dur}(a') \leqslant T(a)$ to Ord_T. The action a' is added to *Steps* if $a' \notin \mathit{Steps}$ and in such case a variable $T(a')$ for a' is created. Similarly, causal link *threats*, i.e., situations in which an action $a \in \mathit{Steps}$ may delete a condition $p \in \mathit{del}(a)$ in a causal link $a_1[p]a_2$ in *CL*, are solved by adding one of the temporal constraints $T(a) + \mathit{dur}(a) \leqslant T(a_1)$ or

$T(a_2) + dur(a_2) \leqslant T(a)$ 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 state*).

The temporal constraints in Ord_T form a Simple Temporal Problem (STP) [9] whose consistency can be tested efficiently by applying a form of constraint propagation known as *bounds consistency* [29,48], where the lower and upper bounds $T_{min}(a)$ and $T_{max}(a)$ of the variables $T(a)$ in constraints of the form $T(a) + dur(a) \leqslant T(a')$ 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 $\sigma = \langle Steps, Ord_T, CL, Open, T(\cdot)\rangle$ encode a valid temporal plan P with makespan B where actions in σ execute at their earliest possible times; i.e., $P = \langle a_i, t_i = T_{min}(a_i)\rangle_{a_i \in Steps}$, and one such terminal goal state will be generated when one such valid temporal 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 $T_{min}(a) + dur(a) \leqslant T_{min}(a_1)$ and $T_{min}(a_2) + dur(a_2) \leqslant T_{min}(a)$ hold. This is because the lower bounds T_{min} provide a consistent solution to a STP if the STP is consistent, and at the same time, each of the constraints $T(a) + dur(a) \leqslant T(a_1)$ and $T(a_2) + dur(a_2) \leqslant T(a)$ posted as a result of a threat fix the threat through bounds consistency propagation. 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 easy to verify that the branching scheme above ensures that precondition-interfering actions cannot overlap in time in the final plan, as such interferences give rise to causal link threats. On the other hand, effect-interfering actions may overlap. To rule out such situations, 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 $T_{min}(a) + dur(a) \leqslant T_{min}(a')$ nor $T_{min}(a') + dur(a') \leqslant T_{min}(a)$ hold in the state σ. Such flaws are solved by adding to Ord_T one of the temporal constraints $T(a) + dur(a) \leqslant T(a')$ or $T(a') + dur(a') \leqslant T(a)$.

Modern Constraint-Based Interval (CBI) planners [18,39] are based on similar ideas and are able to deal with more expressive languages. Yet, as in standard POCL and Dynamic CSP planners [19], 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 eventually become inconsistent* are needed. This is indeed what CPT does in the TOWER-n domain considered above for planning horizons smaller than the optimal horizon, reporting an inconsistency by pure inference without doing any search. Moreover, in the same domain, for the optimal planning horizon, CPT finds the solution without doing any search either. In both cases, as we see next, the key is the ability of CPT to reason about all the actions in the problem, and not only about the actions in the plan being considered.

5. 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 'standard' 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 $T(p,a) = T(a')$.[3]

Initially, we will follow the formulation in [45], and make an important restriction; namely that *no (ground) action a in the domain occurs more than once in the plan.* This *canonicity restriction* allows us to collapse the notions of action and action occurrence, leading to a number of simplifications. Later on we will show how this restriction is removed in the current version of CPT. The restriction is a meaningful extension of the common assumption found in scheduling research where every action in the domain must occur exactly *once,* and as we will see below, it happens to be true in most current benchmarks in planning.

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.

5.1. 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 [17]. The values provide lower bounds on the times to achieve the preconditions of a and the pair of atoms

[3] Propositional 'causal' encodings of Strips planning problems have been formulated and analyzed in [22,31]. Our encodings share a number of features with these formulations but are more compact due to the use of a temporal representation.

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 [35]).

In addition, the simpler heuristic h^1_T is used for defining *distances* between actions [42] 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. These distances are not symmetric and their calculation, which remains polynomial, involves the computation of the h^1_T heuristic $|O|$ times.

The distances $dist(Start, a)$ and $dist(a, End)$ 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, minus $dur(a)$. The distances $dist(Start, a)$ are set to $h^2_T(a)$.

5.2. 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 (i.e., actions for which the domain of $InPlan(a)$ remains the interval $[0, 1]$ in σ) 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.

5.3. Constraints

The constraints correspond basically to disjunctions, rules, and temporal constraints, or their combination. Most of these constraints are redundant; they are not needed for soundness or completeness but for performance reasons (pruning values and detecting inconsistencies earlier). 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 $T_{min}(a) \geqslant T_{max}(a')$ holds, and otherwise is *undetermined*.[4] Temporal constraints are propagated by 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 $a \in O$,

$$T(Start) + \delta(Start, a) \leqslant T(a),$$
$$T(a) + \delta(a, End) \leqslant T(End).$$

- *Preconditions*: Supporter a' of precondition p of a must precede a by an amount that depends on $\delta(a',a)$:

$$T(a) \geqslant \min_{a' \in D[S(p,a)]} \big(T(a') + \delta(a',a)\big),$$
$$T(a) \geqslant T(p,a) + \min_{a' \in D[S(p,a)]} \delta(a',a),$$
$$T(a') + \delta(a',a) > T(a) \rightarrow S(p,a) \neq a'.$$

- *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

$$T(a') + dur(a') + \min_{a'' \in D[S(p,a)]} dist(a', a'')$$
$$\leqslant T(p,a) \vee T(a) + \delta(a,a') \leqslant T(a').$$

- *Mutex constraints*: For effect-interfering a and a'

$$T(a) + \delta(a,a') \leqslant T(a') \vee T(a') + \delta(a',a) \leqslant T(a).$$

[4] 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 $D[S(p,a)] = \{a'\}$ 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 [44]. We also note that $T(a) < T(a')$ is true in our modified CP engine when $a' = End$, regardless of the domain of $T(a)$.

- *Support constraints*: $T(p,a)$ and $S(p,a)$ related by

$$S(p,a) = a' \rightarrow T(p,a) = T(a'),$$
$$T(p,a) \neq T(a') \rightarrow S(p,a) \neq a',$$
$$\min_{a' \in D[S(p,a)]} T(a') \leqslant T(p,a) \leqslant \max_{a' \in D[S(p,a)]} T(a').$$

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 $\mathit{InPlan}(a) = 1$ is neither true or false. They become *in-plan* variables when $\mathit{InPlan}(a) = 1$ becomes true, and *out-plan* variables when $\mathit{InPlan}(a) = 1$ 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 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, $\mathit{InPlan}(a)$ 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, $\mathit{InPlan}(a)$ is automatically set to 1. Conditional variables of this type in constraint programming have been considered in [13].

5.4. Branching

As in the temporal POCL planner described above, branching in CPT proceeds by iteratively selecting and fixing flaws in non-terminal states σ and backtracking upon inconsistencies. A state σ is given by the variables, their domains, and the constraints involving them. The initial state σ_0 contains the variables, domains, and constraints above, along with the bounding constraint $T(\mathit{End}) = B$ where B is the current bound on the makespan. A state is inconsistent when a non-conditional variable has an empty domain, while a consistent state σ with no flaws is a *goal state* from which a valid plan P with bound B can be extracted by scheduling the in-plan variables at their earliest starting times.

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 $T_{min}(a') + \mathit{dur}(a') \leqslant T_{min}(p,a)$ nor $T_{min}(a) + \mathit{dur}(a) \leqslant T_{min}(a')$ 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 $T_{min}(a) + dur(a) \leqslant T_{min}(a')$ nor $T_{min}(a') + dur(a') \leqslant T_{min}(a)$ 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
$$\Big[T(a') + dur(a') + \min_{a'' \in D[S(p,a)]} dist(a', a'') \leqslant T(p,a);$$
$$T(a) + \delta(a, a') \leqslant T(a')\Big].$$
- An *Open Condition* $S(p,a)$ generates for a selected support a' the split
$$\big[S(p,a) = a';\ S(p,a) \neq a'\big].$$
- A *Mutex Threat* $\langle a, a'\rangle$ generates the split
$$\big[T(a) + \delta(a,a') \leqslant T(a');\ T(a') + \delta(a',a) \leqslant T(a)\big].$$

The branching scheme is sound and complete under the canonical restrictions above. 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 binary splits: namely for each possible binary split $[C_1; C_2]$, the disjunction $C_1 \vee C_2$ is valid; thus if there is a plan with makespan B compatible with the commitments in σ, then there will be a plan compatible with one of the two sons of σ.

5.4.1. Branching heuristics

In each step, the selected flaw for repair in CPT is a Support Threat if one exists, else an Open Condition if one exists, else a Mutex Threat, until no flaws are left or an inconsistency is detected. The heuristic for selecting among the existing flaws is the following:

- *Support Threats* $\langle a', S(p,a)\rangle$ with minimum slack
$$\max\big[slack\big(a' \prec S(p,a)\big), slack(a \prec a')\big]$$
selected first (i.e., most constrained first; see [41]). Basically, the slack of an ordering $a \prec a'$ stands for the 'room' for a' in the schedule assuming it must follow a; namely,
$$slack(a \prec a') = T_{max}(a') - \big[T_{min}(a) + \delta(a,a')\big],$$
$$slack\big(a' \prec S(p,a)\big) = T_{max}(p,a) - \Big[T_{min}(a') + dur(a') + \min_{a'' \in D[S(p,a)]} dist(a', a'')\Big].$$

- *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 in simple fashion; first encountered such pair in a search over *Steps* selected first.

The heuristics for Support Threats and Open Conditions have a significant influence on performance but not so the heuristic for Mutex Threats (most often no Mutex Threats are left after removal of Support Threats and Open Conditions).

5.5. Mutex sets

The code incorporates an enhancement that helps in some domains without representing a significant burden 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:

$$\max_{a \in M} \big[T_{max}(a) + dur(a) \big] - \min_{a \in M} T_{min}(a)$$

must provide enough 'room' for scheduling all actions in $a \in M$ in sequence. Taking into account the pre-computed distances, an estimate for the time needed for scheduling all actions in M is given by

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

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

$$\max_{a' \in M} \big[T(a') + dur(a') \big] - \min_{a'' \in M} T(a'') \geqslant \Delta(M)$$

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 [2,6,26].

- *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 $T(a)$:

$$\min_{a' \in M^-} T(a') + \Delta(M^-) \leqslant T(p,a) \ \wedge \ T(a) + dur(a) \leqslant \max_{a' \in M^+} \big[T(a') + dur(a')\big] - \Delta(M^+).$$

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 $S(p,a)$).

5.6. Relaxation of the canonicity assumption

The formulation above 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, making the formulation simpler but less general. In the current CPT planner, this restriction is removed by establishing a distinction between *action types* and *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. On the other hand, constraints and domains, that initially involve only action types, eventually involve *both* action tokens and types. Basically, an action type is regarded as a place holder for all the action tokens of that type that have not made it yet into the plan. Action tokens are created dynamically from action types when an action type is selected for supporting an open condition in the plan. This happens when the propagation narrows down the domain of a support variable $S(p,b)$ for an action (token) b in the plan to the singleton $\{a\}$, where a is an action type, or when the action type a is explicitly chosen as the value of a support variable $S(p,b)$. In such a case, a new token a' of type a is created by 'cloning'; namely for the new instance a' of type a, the variables $T(a')$, $S(q,a')$, and $T(q,a')$ are created as fresh copies of the variables $T(a)$, $S(q,a)$, and $T(q,a)$ with their corresponding domains, where q 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, and all the constraints involving the variables $T(a)$, $S(q,a)$, and $T(q,a)$ are copied with a' in place of a. The value of the variable $\mathit{InPlan}(a')$ is then set to 1 and a' is added to *Steps*. Finally, if the action instance a' of the action type a was created because action type a was chosen (by branching or propagation) to support the precondition p of an action b, then the variable $S(p,b)$ is set to the new instance a' of a.

As an illustration, let us consider a problem in the Blocks World domain with three blocks A, B and C with $on(C,B)$ true in the initial state. The action $stack(A,B)$ has $clear(B)$ as precondition, so the domain of the support variable $S(clear(B),\ stack(A,B))$ is equal to $\{putdown(B),\ stack(B,A),\ stack(B,C),\ unstack(A,B),\ unstack(C,B)\}$. Suppose now that $\mathit{InPlan}(stack(A,B)) = 1$ and that the 'Open Condition' branching rule chooses as the value of the support variable $S(clear(B),\ stack(A,B))$ the action type

unstack(C, B). The 'cloning' operation then creates the new action token *unstack*$(C, B)'$ of type *unstack*(C, B), and then performs the following operations:

- First, the *variables* *InPlan*(*unstack*$(C, B)'$), T(*unstack*$(C, B)'$), S(*clear*(C), *unstack*$(C, B)'$), S(*on*(C, B), *unstack*$(C, B)'$), T(*clear*(C), *unstack*$(C, B)'$) and T(*on*(C, B), *unstack*$(C, B)'$) are created, their domains being a copy of the corresponding domains of the variables involving the action type *unstack*(C, B). For instance, if the domain of the temporal variable T(*unstack*(C, B)) is $[0, 5]$, then the domain of the cloned variable T(*unstack*$(C, B)'$) is set to $[0, 5]$ as well.
- Then all the *constraints* involving the type *unstack*(C, B) are copied with the token *unstack*$(C, B)'$ instead of *unstack*(C, B), and all these constraints are entered into the current state. For example, the following new precondition constraints are added

$$T\big(\mathit{unstack}(C, B)'\big) \geqslant \min_{a' \in D[S(\mathit{clear}(B), \mathit{unstack}(C, B)')]} \big(T(a') + \delta\big(a', \mathit{unstack}(C, B)'\big)\big)$$

and

$$T\big(\mathit{unstack}(C, B)'\big) \geqslant \min_{a' \in D[S(\mathit{on}(C, B), \mathit{unstack}(C, B)')]} \big(T(a') + \delta\big(a', \mathit{unstack}(C, B)'\big)\big).$$

- Also the *domains* of all the support variables containing the action type *unstack*(C, B) are extended with the new action token *unstack*$(C, B)'$. For example, since *unstack*(C, B) produces *holding*(C), the domain of S(*holding*(C), *stack*(C, A)) which was equal to {*pickup*(C), *unstack*(C, A), *unstack*(C, B)} is augmented with *unstack*$(C, B)'$; i.e., $D[S(\mathit{holding}(C), \mathit{stack}(C, A))]$ becomes equal to {*pickup*(C), *unstack*(C, A), *unstack*(C, B), *unstack*$(C, B)'$}. Similarly, *unstack*$(C, B)'$ is added to $D[S(\mathit{clear}(B), \mathit{pickup}(B))]$, which becomes equal to {*unstack*(A, B), *unstack*(C, B), *unstack*$(C, B)'$}.
- Finally, the *causal link is instantiated*; i.e., the support variable S(*clear*(B), *stack*(A, B)) is set to the new token *unstack*$(C, B)'$ which is added to the plan by setting *InPlan*(*unstack*$(C, B)'$) to 1, and the effects are propagated.

This scheme provides a lazy implementation of a planning domain with an infinite number of action tokens. In such a scheme, an action type represents all the action instances of that type that have not made it yet into the plan, and which are thus indistinguishable up to that point. This changes however when a new instance is added to the plan, requiring the 'cloning' operation detailed above. In our example, after the action token *unstack*$(C, B)'$ is 'cloned' from the action type *unstack*(C, B), the two actions become 'independent', meaning that from that point on, things work as if they were two completely different actions in the domain.

Notice that if during the search *InPlan*$(a) = 0$ for an action type a is inferred, all new action tokens of that type get automatically excluded from the plan. Namely, action types are true place holders for the information that is common to all the action tokens of the same type that are not yet in the plan.

5.7. Implementation

The CPT planner has been implemented using the CHOCO CP library [28] that operates on top of the CLAIRE programming language [7] and 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 corresponding propagations (namely, updates on variables constrained by the modified variables are done which may trigger other rules and further updates). 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 for allowing to retract such constraints upon backtracking, 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. Over temporal variables, the conditional behavior is obtained by handling those variables ourselves, while over support variables, the conditional behavior is obtained by simply introducing a dummy action α added to their domains, with $D[S(p,a)] = \{\alpha\}$ meaning that p cannot be supported by any action. The $\mathit{InPlan}(a)$ 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. Finally, for the removal of the canonicity restriction, the CHOCO engine was extended so that variables can be created dynamically, values can be added dynamically to their domains, and all such actions can be retracted upon backtracking. The code and several executables are available for download from our page.[5]

6. A working example

We revisit the example in Section 2 for showing how the backtrack-free behavior of CPT in the TOWER-n domain follows from the proposed constraint programming formulation. Recall that the task in TOWER-n is to build an ordered tower of n blocks, $b_1, \ldots, b_n$, with b_1 on top, all blocks laying initially 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}, from $i = n-1$ until $i = 1$. This is a trivial domain but which no other optimal planner solves without search. Indeed, the inferences are not trivial for a domain-independent planner as we will see.

The *temporal variables* and their domains after preprocessing are ($i, j \in [1, n], i \neq j$):

[5] CPT home page: http://www.cril.univ-artois.fr/~vidal/cpt.en.html.

- $T(Start) :: [0, \infty]$
- $T(End) :: [4, \infty]$
- $T(pickup(b_i)) :: [0, \infty]$
- $T(putdown(b_i)) :: [1, \infty]$
- $T(stack(b_i, b_j)) :: [1, \infty]$
- $T(unstack(b_i, b_j)) :: [2, \infty]$
- $T(on(b_i, b_{i+1}), End) :: [1, \infty]$
- $T(ontable(b_i), pickup(b_i)) :: [0, \infty]$
- $T(handempty, pickup(b_i)) :: [0, \infty]$
- $T(clear(b_i), pickup(b_i)) :: [0, \infty]$
- $T(holding(b_i), putdown(b_i)) :: [0, \infty]$
- $T(on(b_i, b_j), unstack(b_i, b_j)) :: [1, \infty]$
- $T(handempty, unstack(b_i, b_j)) :: [0, \infty]$
- $T(clear(b_i), unstack(b_i, b_j)) :: [0, \infty]$
- $T(holding(b_i), stack(b_i, b_j)) :: [0, \infty]$
- $T(clear(b_j), stack(b_i, b_j)) :: [0, \infty]$

The *support variables* and their domains in turn are:

- $S(on(b_i, b_{i+1}), End) :: \{stack(b_i, b_{i+1})\}$
- $S(ontable(b_i), pickup(b_i)) :: \{Start, putdown(b_i)\}$
- $S(handempty, pickup(b_i)) :: \{Start\} \cup PUTDOWN \cup STACK$
- $S(clear(b_i), pickup(b_i)) :: \{Start, putdown(b_i)\} \cup STACK_{i,*} \cup UNSTACK_{*,i}$
- $S(holding(b_i), putdown(b_i)) :: \{pickup(b_i)\} \cup UNSTACK_{i,*}$
- $S(on(b_i, b_j), unstack(b_i, b_j)) :: \{stack(b_i, b_j)\}$
- $S(handempty, unstack(b_i, b_j)) :: \{Start\} \cup PUTDOWN \cup STACK$
- $S(clear(b_i), unstack(b_i, b_j)) :: \{Start, putdown(b_i)\} \cup STACK_{i,*} \cup UNSTACK_{*,i}$
- $S(holding(b_i), stack(b_i, b_j)) :: \{pickup(b_i)\} \cup UNSTACK_{i,*}$
- $S(clear(b_j), stack(b_i, b_j)) :: \{Start, putdown(b_j)\} \cup STACK_{j,*} \cup UNSTACK_{*,j}$

where

- $PICKUP = \{pickup(b_i) \mid i \in [1, n]\}$
- $PUTDOWN = \{putdown(b_i) \mid i \in [1, n]\}$
- $STACK = \{stack(b_i, b_j) \mid i, j \in [1, n] \wedge j \neq i\}$
- $STACK_{i,*} = \{stack(b_i, b_j) \mid j \in [1, n] \wedge j \neq i\}$
- $STACK_{*,i} = \{stack(b_j, b_i) \mid j \in [1, n] \wedge j \neq i\}$
- $UNSTACK = \{unstack(b_i, b_j) \mid i, j \in [1, n] \wedge j \neq i\}$
- $UNSTACK_{i,*} = \{unstack(b_i, b_j) \mid j \in [1, n] \wedge j \neq i\}$
- $UNSTACK_{*,i} = \{unstack(b_j, b_i) \mid j \in [1, n] \wedge j \neq i\}$

We explain the inferences that yield the backtrack-free behavior in TOWER-n by quoting the high-level account in Section 2, and showing how it follows from the constraints in CPT and the general constraint propagation mechanisms supported in the implementation. For

keeping the description simple we describe the canonical implementation where there is no need for distinguishing action types from tokens.

Step 1: Addition of *stack* actions to the plan.

...First, it is inferred that each subgoal $on(b_i, b_{i+1})$ must be achieved by the action $stack(b_i, b_{i+1})$. This inference is simple as there is a single possible supporter in each case ...

- For each $i \in [1, n-1]$ indeed, $S(on(b_i, b_{i+1}), End)$ has a singleton domain, and since $InPlan(End) = 1$, $S(on(b_i, b_{i+1}), End) = stack(b_i, b_{i+1})$ and $InPlan(stack(b_i, b_{i+1})) = 1$ are inferred.

Step 2: Increasing the starting times of *stack* actions.

...More interestingly, it is then inferred that these stack operations must be ordered sequentially in descending order of i; namely, $stack(b_{n-1}, b_n)$ first, then $stack(b_{n-2}, b_{n-1})$, and so on, until $stack(b_1, b_2)$. This is inferred by reasoning with and resolving the threats affecting the causal links $stack(b_i, b_{i+1})[on(b_i, b_{i+1})]End$. Moreover, it is also inferred that the first action in the sequence cannot occur earlier than $t = 1$, the second action not earlier than $t = 3$, the third not earlier than $t = 5$, and so on, and that the *End* action cannot start earlier than $2(n-1)$, the optimal time bound ...

- The action $stack(b_{n-1}, b_n)$ e-deletes $on(b_{n-2}, b_{n-1})$, and so threatens the causal link $stack(b_{n-2}, b_{n-1})[on(b_{n-2}, b_{n-1})]End$. Following the causal link constraint, since $stack(b_{n-1}, b_n)$ cannot follow *End*, it must precede $stack(b_{n-2}, b_{n-1})$, and hence the disjunct

$$T(a') + dur(a') + \min_{a'' \in D[S(p,a)]} dist(a', a'') \leqslant T(p, a)$$

  with $p = on(b_{n-2}, b_{n-1})$, $a = End$ and $a' = stack(b_{n-2}, b_{n-1})$ is inferred, which since $dist(stack(b_{n-1}, b_n), stack(b_{n-2}, b_{n-1})) = 1$ and $dur(stack(b_{n-1}, b_n)) = 1$, yields

$$T\big(stack(b_{n-1}, b_n)\big) + 2 \leqslant T\big(on(b_{n-2}, b_{n-1}), End\big)$$

  and therefore

$$T\big(on(b_{n-2}, b_{n-1}), End\big) \geqslant 3$$

  as from preprocessing, $T(stack(b_i, b_j)) \geqslant 1$ for all i, j.
- Then from the constraint $S(p, a) = a' \rightarrow T(p, a) = T(a')$ and the inferred constraint $S(on(b_{n-2}, b_{n-1}), End) = stack(b_{n-2}, b_{n-1})$,

$$T\big(stack(b_{n-2}, b_{n-1})\big) \geqslant 3.$$

- In a similar way, the disjunct

$$T\big(stack(b_{n-2}, b_{n-1})\big) + 2 \leqslant T\big(on(b_{n-3}, b_{n-2}), End\big)$$

  of the causal link constraint becomes active, and since $T(stack(b_{n-2}, b_{n-1})) \geqslant 3$ holds, so does $T(on(b_{n-3}, b_{n-2}), End) \geqslant 5$, and from the constraint $S(p, a) = a' \rightarrow T(p, a) = T(a')$ and $S(on(b_{n-3}, b_{n-2}), End) = stack(b_{n-3}, b_{n-2})$,

$$T\big(stack(b_{n-3}, b_{n-2})\big) \geqslant 5.$$

- The same process is iterated over all the actions $stack(b_i, b_{i+1})$ until

  $$T\big(stack(b_1, b_2)\big) \geqslant 2(n-1) - 1.$$

  Then, as $S(on(b_1, b_2), End) = stack(b_1, b_2)$, the precondition constraint

  $$T(a) \geqslant \min_{a' \in D[S(p,a)]} \big(T(a') + \delta(a', a)\big)$$

  for $a = End$ and $p = on(b_1, b_2)$, results in

  $$T(End) \geqslant T\big(stack(b_1, b_2)\big) + 1$$

  which from $T(stack(b_1, b_2)) \geqslant 2(n-1) - 1$, yields $T(End) \geqslant 2(n-1)$.

Step 3: Setting the initial upper bound on the makespan and deriving upper bounds for the *stack* actions.

...All these inferences result from the domain constraints and propagation mechanisms before even a search bound B on the allowed makespan of the plan is fixed. After the first bound $B = 2(n-1)$ is chosen (this is the earliest time at which the action *End* can start), further inferences are made. First, the starting times $T(a_k)$ of all the actions a_i in the stack sequence above become fixed to their earliest possible starting times resulting in $T(a_k) = 1 + 2k$, for $k = 1, \ldots, n-1$, where a_k is the kth action in the sequence (namely $a_k = stack(b_{n-k}, b_{n-k+1})$)...

- The constraint $T(End) = B$ on the makespan is asserted for B equal to the current lower bound $2(n-1)$ of variable $T(End)$, and then from the bounding constraint

  $$T(a) + \delta(a, End) \leqslant T(End)$$

  for $a = stack(b_1, b_2)$, and $\delta(stack(b_1, b_2), End) = 1$ (the *stack* actions have duration 1), it is inferred that

  $$T\big(stack(b_1, b_2)\big) \leqslant 2(n-1) - 1$$

  and since we have $T(stack(b_1, b_2)) \geqslant 2(n-1) - 1$, that

  $$T\big(stack(b_1, b_2)\big) = 2(n-1) - 1.$$

- From the constraint $S(p, a) = a' \rightarrow T(p, a) = T(a')$, in turn, and $S(on(b_1, b_2), End) = stack(b_1, b_2)$, it is inferred also that

  $$T\big(on(b_1, b_2), End\big) = 2(n-1) - 1.$$

- Then from the constraint $T(stack(b_2, b_3)) + 2 \leqslant T(on(b_1, b_2), End)$ derived in step 2, this propagates into

  $$T\big(stack(b_2, b_3)\big) \leqslant 2(n-1) - 3$$

  but since $T(stack(b_2, b_3)) \geqslant 2(n-1) - 3$ also from step 2, then

  $$T\big(stack(b_2, b_3)\big) = 2(n-1) - 3.$$

- This continues iteratively until obtaining

  $$T\big(stack(b_{n-1}, b_n)\big) = 1.$$

Step 4: Addition of $pickup(b_{n-1})$ to the plan.

...Then the $pickup(b_{n-1})$, $pickup(b_{n-2})$, ...sequence gets added to the set of actions in the plan at their correct starting times as a result of further reasoning that prunes the other possible supports and times. For example, the precondition $clear(b_n)$ for the first action $a_1 = stack(b_{n-1}, b_n)$ in the sequence can be supported by a number of $unstack(*, b_n)$ and $stack(b_n, *)$ actions, and by *Start*. However, since any such supporter a' must precede a_1 and $T(a_1) = 1$ is already fixed, $T(a') < 1$ must hold, leaving $a' = Start$ as the only possible supporter *(*at preprocessing, lower bounds on the starting time of actions are computed from which it is known that $T(a') < 1$ is true only for *Start* and *pickup* actions*)*. For similar reasons, all supporters $unstack(b_{n-1}, *)$ for the other precondition $holding(b_{n-1})$ of a_1 are pruned, leaving $a'_1 = pickup(b_{n-1})$ as the only possible support. The process repeats for the preconditions of $a'_1 = pickup(b_{n-1})$ with all supporters a' different than *Start* being pruned as well ...

- $stack(b_{n-1}, b_n)$ has two preconditions: $clear(b_n)$ and $holding(b_{n-1})$. From the constraint $T(a) \geqslant T(p, a) + \min_{a' \in D[S(p,a)]} \delta(a', a)$ with $p = clear(b_n)$ and $a = stack(b_{n-1}, b_n)$, as $T(stack(b_{n-1}, b_n)) = 1$, it is inferred that $T(clear(b_n), stack(b_{n-1}, b_n)) \leqslant 0$ and hence that

  $$T\big(clear(b_n), stack(b_{n-1}, b_n)\big) = 0.$$

- The domain of variable $S(clear(b_n), stack(b_{n-1}, b_n))$ contains *Start* and the actions in $STACK_{n,*}$ and $UNSTACK_{*,n}$. However, from preprocessing, the actions in $STACK_{n,*}$ have starting times greater than or equal to 1, and the actions in $UNSTACK_{*,n}$ have starting times greater than or equal to 2. From the constraint

  $$T(p, a) \neq T(a') \rightarrow S(p, a) \neq a'$$

  with $p = clear(b_n)$, $a = stack(b_{n-1}, b_n)$ and $a' \in STACK_{n,*} \cup UNSTACK_{*,n}$, all the actions in $STACK_{n,*}$ and $UNSTACK_{n-1,*}$ are then pruned from the domain of the variable $S(clear(b_n), stack(b_{n-1}, b_n))$. The only remaining action is then *Start*, and we have then

  $$S\big(clear(b_n), stack(b_{n-1}, b_n)\big) = Start.$$

- For the second precondition of $stack(b_{n-1}, b_n)$, i.e., $holding(b_{n-1})$, the reasoning is similar: first $T(holding(b_{n-1}), stack(b_{n-1}, b_n)) = 0$ is inferred, and then since $holding(b_{n-1})$ can be produced only by $pickup(b_{n-1})$ and the actions $UNSTACK_{n-1,*}$ which all have starting times greater than or equal to 2, it follows from $T(p, a) \neq T(a') \rightarrow S(p, a) \neq a'$ with $p = holding(b_{n-1})$, $a = stack(b_{n-1}, b_n)$ and $a' \in UNSTACK_{n-1,*}$, that all such actions a' are pruned from $D[S(holding(b_{n-1}), stack(b_{n-1}, b_n))]$, resulting in

  $$S\big(holding(b_{n-1}), stack(b_{n-1}, b_n)\big) = pickup(b_{n-1})$$

  and

  $$InPlan\big(pickup(b_{n-1})\big) = 1.$$

- Furthermore, from the constraint $S(p,a) = a' \rightarrow T(p,a) = T(a')$ it is also inferred that $T(\mathit{pickup}(b_{n-1})) = 0$, and from the precondition constraint

$$T(a) \geqslant T(p,a) + \min_{a' \in D[S(p,a)]} \delta(a',a)$$

and $a = \mathit{pickup}(b_{n-1})$, $T(p,a) = 0$ is inferred for the two preconditions p of a: $\mathit{clear}(b_{n-1})$ and *handempty*. As a result, from the constraint $T(p,a) \neq T(a') \rightarrow S(p,a) \neq a'$, all actions other than *Start* are pruned as possible supporters of $\mathit{clear}(b_{n-1})$ and *handempty*, from which it is inferred that

$$S\big(\mathit{clear}(b_{n-1}), \mathit{pickup}(b_{n-1})\big) = S\big(\mathit{handempty}, \mathit{pickup}(b_{n-1})\big) = \mathit{Start}.$$

Step 5: Addition of $\mathit{pickup}(b_{n-2})$ to the plan.

... At this point a number of actions and causal links in the plan have been inferred with no commitments made except for the bound B. In particular, due to the causal links going into the actions $\mathit{pickup}(b_{n-1})$ and $\mathit{stack}(b_{n-1}, b_n)$ already fixed at the times $t = 0$ and $t = 1$ respectively, and the fact that all actions a' whether in the plan or not (except for these two and *Start*), threat these causal links but cannot precede both actions, the starting times $T(a')$ of such actions a' are pushed to times $t = 2$ or higher. The result is that the only supporters left for the preconditions $\mathit{clear}(b_{n-1})$ and $\mathit{holding}(b_{n-2})$ of the next stack action in the sequence, $a_2 = \mathit{stack}(b_{n-2}, b_{n-1})$, scheduled at time $t = 3$, end up being the actions $a_1 = \mathit{stack}(b_{n-1}, b_n)$ at $t = 1$ and $\mathit{pickup}(b_{n-2})$ at time $t = 2$ ...

- The action $\mathit{stack}(b_{n-2}, b_{n-1})$ still has two open preconditions: $\mathit{holding}(b_{n-2})$ and $\mathit{clear}(b_{n-1})$. The action $\mathit{stack}(b_{n-1}, b_n)$ e-deletes $\mathit{holding}(b_{n-2})$, and thus threats the support variable $S(\mathit{holding}(b_{n-2}), \mathit{stack}(b_{n-2}, b_{n-1}))$. But since it does not precede $\mathit{stack}(b_{n-2}, b_{n-1})$ (all the times for the *stack* actions are already fixed), the first disjunct of the causal link constraint is enforced

$$T(a') + \mathit{dur}(a') + \min_{a'' \in D[S(p,a)]} \mathit{dist}(a', a'') \leqslant T(p,a)$$

with $p = \mathit{holding}(b_{n-2})$, $a' = \mathit{stack}(b_{n-1}, b_n)$ and $a = \mathit{stack}(b_{n-2}, b_{n-1})$ which yields

$$T\big(\mathit{holding}(b_{n-2}), \mathit{stack}(b_{n-2}, b_{n-1})\big) \geqslant 2.$$

In turn from $T(p,a) + \min_{a' \in D[S(p,a)]} \delta(a',a) \leqslant T(a)$ with $p = \mathit{holding}(b_{n-2})$ and $a = \mathit{stack}(b_{n-2}, b_{n-1})$, $T(\mathit{holding}(b_{n-2}), \mathit{stack}(b_{n-2}, b_{n-1})) \geqslant 2$ is inferred, and therefore from the inequality above,

$$T\big(\mathit{holding}(b_{n-2}), \mathit{stack}(b_{n-2}, b_{n-1})\big) = 2.$$

- The actions that can support the precondition $\mathit{holding}(b_{n-2})$ of $\mathit{stack}(b_{n-2}, b_{n-1})$ are $\mathit{pickup}(b_{n-2})$ and the actions $\mathit{UNSTACK}_{n-2,*}$. However, the latter actions are excluded. Indeed, they all have as precondition the fact that b_{n-2} is on another block, and the actions that can produce this precondition are the ones in $\mathit{STACK}_{n-2,*}$. However, these actions cannot precede the action $\mathit{stack}(b_{n-1}, b_n)$, which is in the plan, and hence must follow it because of the causal link constraint. Since the distance

between $stack(b_{n-1}, b_n)$ and the actions in $STACK_{n-2,*}$ is 1, the lower bound of the starting time of these actions is increased to 3. As a consequence, the lower bound of the actions in $UNSTACK_{n-2,*}$ is increased to 4, and this is why they cannot produce the precondition $holding(b_{n-2})$ for $stack(b_{n-2}, b_{n-1})$, and therefore

$$S\big(holding(b_{n-2}), stack(b_{n-2}, b_{n-1})\big) = pickup(b_{n-2}).$$

- The actions that can produce the other precondition $clear(b_{n-1})$ of $stack(b_{n-2}, b_{n-1})$ are either *Start*, or the actions in $STACK_{n-1,*} \cup UNSTACK_{*,n-1}$. As $clear(b_{n-1})$ is false before doing $stack(b_{n-1}, b_n)$ and no action is left between $stack(b_{n-1}, b_n)$ and $stack(b_{n-2}, b_{n-1})$, the only possibility is

$$S\big(clear(b_{n-1}), stack(b_{n-2}, b_{n-1})\big) = stack(b_{n-1}, b_n).$$

- The same kind of reasoning is made for the preconditions of $pickup(b_{n-2})$, and therefore the support variables get the values

$$S\big(handempty, pickup(b_{n-2})\big) = stack(b_{n-1}, b_n)$$

and

$$S\big(clear(b_{n-2}), pickup(b_{n-2})\big) = Start.$$

Step 6: Addition of all other *pickup* actions to the plan.

...the process repeats for all other stack actions in the sequence until all actions have their start times and supporters fixed and no flaw in the plan is left.

- Following the same process, the actions in $UNSTACK_{n-k,*}$ with $k \geqslant 3$ are excluded from the domain of the support variables $S(holding(b_{n-k}),\ stack(b_{n-k},\ b_{n-k-1}))$, leaving as the only possible choice the actions $pickup(b_{n-k})$ whose correct starting times are also inferred. The preconditions of the actions $pickup(b_{n-k})$ are found in the same way.

7. Experimental results

We consider next the experiments for comparing CPT with other optimal parallel and temporal planners. The experiments have been obtained using a Pentium IV machine running at 2.8 Ghz, with 1 Gb of RAM, under Linux, and a time limit of one hour for each problem. The planners are:

- CPT: our temporal planner, a version that slightly improves the version entered at the 4th International Planning Competition (Optimal Track; see [11]) with no canonicity restrictions,[6]

[6] While CPT was entered at the 4th IPC, CPT does not adhere completely to the PDDL2.1 semantics [14] but rather follows the simpler semantics for temporal planning in [40]. In the former, plans with smaller makespans may result as interfering actions are allowed to overlap in certain cases. See [14] for details.

- BLACKBOX: the SAT-based parallel planner described in [23] with the CHAFF SAT solver [34],
- SATPLAN04: the new implementation of BLACKBOX with the SIEGE SAT solver, as it was entered at the 4th International Planning Competition,
- IPP: the GRAPHPLAN-based parallel planner described in [24], and
- TP4'04: the new implementation of the temporal planner described in [17], that was also entered at the 4th IPC.

We evaluated the two temporal planners CPT and TP4'04 over temporal domains, and all temporal and parallel planners over parallel domains. The domains and problems are Blocks World (5 standard instances, 50 instances from IPC2), Logistics (8 standard instances, 50 instances from IPC2), Miconic [25] (50 instances from IPC2), and four domains created for IPC3: Depots, DriverLog, Satellite and ZenoTravel. These last four domains are used in both parallel and temporal settings. Details on IPC2 and IPC3 can be found in [1] and [30]. We report results over many domains and instances both for assessing the proposed planner reliably and as a reference for other researchers.

Tables 2–5 compare the planners over the parallel domains, while Table 6 compares CPT and TP4'04 over the temporal domains. The times in all cases include preprocessing. Times reported as 0.00 mean that they were solved in less than 0.01 seconds. The tables show that CPT runtimes and coverage are similar to those of BLACKBOX and SATPLAN04 over the parallel domains with the exception of Blocks World, where CPT does much better, and Logistics and Miconic, where CPT does worse. CPT also seems to scale up much better than IPP over all domains with the exception of the Miconic domain, where IPP does better. Finally, CPT seems to dominate the temporal planner TP4'04 over all parallel and temporal domains, expanding much fewer nodes. As discussed in [17], the problem with state-based temporal planners such as TP4'04 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. While solutions in such a case, may lay deeper in the search tree, pruning decisions have a chance then to prune larger parts of the search space, and therefore, to be more effective.

The scatter plots in Figs. 1–5 summarize the information provided in these tables. The first four figures summarize the results for parallel planning comparing CPT with BLACKBOX, SATPLAN04, IPP and TP4'04 respectively, while the last figure compares CPT with TP4'04 over temporal domains. In these figures, dots represent for each problem, the runtime of CPT (x-axis) in comparison with the runtime of the other planners (y-axis). Dots above the diagonal indicate problems where CPT is faster, while dots below the diagonal indicate problems where the other planners are faster. Likewise, problems on the right border are unsolved by CPT, while problems on the top border are unsolved by the other planners.

The results shown in the tables and in the figures lend support to our main goal in the development of CPT: an optimal temporal planner with good performance, able to approach the performance of the best parallel planners when all actions have the same duration. The key for this result is the combination in CPT of a POCL branching scheme suitable for temporal planning, and a CP representation of partial plans that supports powerful pruning and reasoning mechanisms such as those found in modern parallel planners.

Table 2
Results for Blocks World

	CPU time (sec.)					Makespan
	CPT	BLACKBOX	SATPLAN	IPP	TP4	
bw-12step	0.10	0.15	0.53	0.01	0.17	12
bw-large.a	0.10	0.64	3.35	0.03	0.93	12
bw-large.b	1.02	10.14	181.61	1.33	593.85	18
bw-large.c	140.30	–	–	–	–	28
bw-large.d	–	–	–	–	–	–
bw-ipc01	0.00	0.02	0.17	0.00	0.01	6
bw-ipc02	0.01	0.01	0.20	0.00	0.00	10
bw-ipc03	0.01	0.01	0.16	0.00	0.01	6
bw-ipc04	0.03	0.07	0.34	0.00	0.04	12
bw-ipc05	0.01	0.06	0.32	0.00	0.04	10
bw-ipc06	0.02	0.11	1.28	0.00	0.03	16
bw-ipc07	0.04	0.15	0.48	0.00	0.07	12
bw-ipc08	0.02	0.17	0.90	0.00	0.08	10
bw-ipc09	0.03	0.41	35.14	0.00	0.11	20
bw-ipc10	0.04	0.38	5.07	0.01	0.14	20
bw-ipc11	26.63	2.87	541.39	0.02	3.77	22
bw-ipc12	1.21	1.19	115.31	0.01	0.69	20
bw-ipc13	0.16	2.35	193.12	0.02	2.64	18
bw-ipc14	0.82	3.04	683.88	0.03	5.57	20
bw-ipc15	0.10	1.03	24.37	0.01	0.44	16
bw-ipc16	0.24	6.13	–	0.12	33.03	30
bw-ipc17	0.95	3.35	–	0.04	3.85	28
bw-ipc18	0.12	3.18	–	0.03	1.76	26
bw-ipc19	0.23	12.17	–	0.27	94.41	34
bw-ipc20	1018.47	53.48	–	10.31	–	32
bw-ipc21	16.51	19.93	–	0.71	261.37	34
bw-ipc22	–	75.89	–	9.43	–	32
bw-ipc23	–	283.26	–	390.26	–	30
bw-ipc24	1.11	36.89	–	3.97	2518.01	34
bw-ipc25	574.46	70.49	–	1.86	2936.43	34
bw-ipc26	5.86	39.30	–	0.89	413.27	34
bw-ipc27	0.82	119.58	–	477.50	–	42
bw-ipc28	6.43	198.49	–	281.91	–	44
bw-ipc29	–	–	–	195.42	–	38
bw-ipc30	–	–	–	–	–	–
bw-ipc31	1434.88	–	–	–	–	40
bw-ipc32	6.57	–	–	–	–	52
bw-ipc33	–	–	–	–	–	–
bw-ipc34	1706.01	–	–	–	–	52
bw-ipc35	–	–	–	–	–	–
bw-ipc36	–	–	–	–	–	–
bw-ipc37	–	–	–	–	–	–
bw-ipc38	–	–	–	–	–	–
bw-ipc39	34.15	–	–	–	–	62
bw-ipc40	358.65	–	–	–	–	58
bw-ipc41	–	–	–	–	–	–
bw-ipc42	170.45	–	–	–	–	72

(*continued on next page*)

Table 2 (*continued*)

	CPU time (sec.)					Makespan
	CPT	BLACKBOX	SATPLAN	IPP	TP4	
bw-ipc43	16.86	–	–	–	–	78
bw-ipc44	1563.39	–	–	–	–	68
bw-ipc45	–	–	–	–	–	–
bw-ipc46	–	–	–	–	–	–
bw-ipc47	–	–	–	–	–	–
bw-ipc48	–	–	–	–	–	–
bw-ipc49	–	–	–	–	–	–
bw-ipc50	–	–	–	–	–	–

Table 3
Results for Logistics

	CPU time (sec.)					Makespan
	CPT	BLACKBOX	SATPLAN	IPP	TP4	
log.easy	0.02	0.05	0.19	0.00	0.48	9
rocket.a	0.11	0.22	1.91	4.54	–	7
rocket.b	0.08	0.26	2.68	6.92	–	7
log.a	0.15	0.26	1.05	450.47	–	11
log.b	1.85	0.52	57.92	1190.54	–	13
log.c	2.22	0.85	36.18	–	–	13
log.d	2.82	2.12	100.45	–	–	14
log.d3	1.25	1.69	27.83	–	–	13
log.d1	–	2.77	353.34	–	–	17
log-ipc01	0.02	0.04	0.14	0.00	0.06	9
log-ipc02	0.02	0.04	0.16	0.00	0.07	9
log-ipc03	0.02	0.04	0.17	0.00	0.06	9
log-ipc04	0.02	0.04	0.17	0.00	0.08	9
log-ipc05	0.02	0.04	0.16	0.00	0.10	9
log-ipc06	0.01	0.01	0.11	0.00	0.06	3
log-ipc07	0.02	0.04	0.16	0.00	0.09	9
log-ipc08	0.02	0.04	0.18	0.00	0.89	9
log-ipc09	0.02	0.04	0.20	0.00	0.30	9
log-ipc10	0.09	0.17	0.35	1.32	789.17	12
log-ipc11	0.13	0.22	0.41	24.18	–	13
log-ipc12	0.07	0.15	0.25	0.28	218.13	11
log-ipc13	0.11	0.18	0.30	1.17	1712.18	12
log-ipc14	0.07	0.14	0.29	0.06	30.60	11
log-ipc15	0.07	0.11	0.22	0.02	2.18	10
log-ipc16	1.56	1.37	6.24	–	–	15
log-ipc17	0.21	0.38	1.36	620.86	–	12
log-ipc18	0.43	0.54	1.42	–	–	13
log-ipc19	3.06	1.24	9.94	–	–	15
log-ipc20	0.22	0.48	1.25	–	–	12
log-ipc21	11.39	1.16	8.93	–	–	15
log-ipc22	51.12	2.78	222.94	–	–	13
log-ipc23	2.60	2.14	200.84	–	–	13
log-ipc24	2.36	1.51	72.50	–	–	12

(*continued on next page*)

Table 3 (*continued*)

	CPU time (sec.)					Makespan
	CPT	BLACKBOX	SATPLAN	IPP	TP4	
log-ipc25	2.56	2.67	160.74	–	–	13
log-ipc26	2.50	5.56	182.85	–	–	13
log-ipc27	29.54	1.91	74.16	–	–	12
log-ipc28	6.28	6.88	319.49	–	–	13
log-ipc29	–	10.45	436.39	–	–	13
log-ipc30	1505.16	15.38	591.54	–	–	14
log-ipc31	–	52.05	595.15	–	–	14
log-ipc32	–	90.98	919.45	–	–	15
log-ipc33	1298.66	6.20	326.12	–	–	13
log-ipc34	–	271.41	885.38	–	–	15
log-ipc35	–	26.46	496.10	–	–	14
log-ipc36	–	845.25	924.73	–	–	15
log-ipc37	–	3308.84	–	–	–	16
log-ipc38	–	–	–	–	–	–
log-ipc39	–	84.66	1267.65	–	–	14
log-ipc40	–	–	–	–	–	–

Table 4
Results for Miconic

	CPU time (sec.)					Makespan
	CPT	BLACKBOX	SATPLAN	IPP	TP4	
miconic01	0.00	0.00	0.14	0.00	0.00	4
miconic02	0.00	0.00	0.14	0.00	0.00	3
miconic03	0.00	0.00	0.13	0.00	0.00	4
miconic04	0.00	0.00	0.13	0.00	0.00	4
miconic05	0.00	0.00	0.14	0.00	0.00	4
miconic06	0.00	0.01	0.16	0.00	0.00	6
miconic07	0.00	0.01	0.16	0.00	0.00	6
miconic08	0.00	0.01	0.15	0.00	0.00	6
miconic09	0.00	0.01	0.15	0.00	0.01	6
miconic10	0.00	0.01	0.16	0.00	0.00	6
miconic11	0.01	0.05	1.28	0.00	0.02	8
miconic12	0.01	0.07	26.91	0.00	0.03	10
miconic13	0.01	0.04	0.43	0.00	0.05	8
miconic14	0.01	0.06	11.73	0.00	0.02	9
miconic15	0.01	0.05	0.84	0.00	0.04	8
miconic16	0.02	0.34	228.80	0.00	3.94	12
miconic17	0.02	0.33	143.00	0.00	3.01	11
miconic18	0.11	0.88	444.54	0.00	88.08	14
miconic19	0.14	0.84	403.35	0.01	88.45	14
miconic20	0.18	0.87	459.46	0.00	129.61	14
miconic21	0.35	2.35	377.01	0.03	1054.36	14
miconic22	0.81	4.50	450.55	0.05	–	15
miconic23	0.05	0.51	107.62	0.00	1.52	10
miconic24	0.16	3.31	350.99	0.04	921.91	14
miconic25	0.04	3.79	574.59	0.03	–	16

(*continued on next page*)

Table 4 (*continued*)

	CPU time (sec.)					Makespan
	CPT	BLACKBOX	SATPLAN	IPP	TP4	
miconic26	0.71	4.04	353.35	0.13	–	14
miconic27	0.08	5.12	438.61	0.11	–	15
miconic28	2.38	20.63	506.44	0.21	–	16
miconic29	2.63	17.19	549.54	0.14	–	16
miconic30	30.69	42.18	765.94	0.21	–	18
miconic31	6.89	69.74	808.73	0.88	–	18
miconic32	339.13	150.71	1254.80	1.27	–	20
miconic33	27.82	20.63	676.15	0.66	–	17
miconic34	45.48	33.91	697.31	0.86	–	17
miconic35	0.13	1149.07	1964.92	1.52	–	23
miconic36	8.02	398.61	1717.08	6.26	–	22
miconic37	–	1702.56	–	6.76	–	23
miconic38	958.34	148.81	1273.84	5.44	–	20
miconic39	–	1802.74	2173.08	7.08	–	24
miconic40	–	504.42	1598.66	6.53	–	22
miconic41	–	–	–	34.64	–	26
miconic42	28.24	2240.95	–	29.55	–	24
miconic43	0.32	2317.58	–	34.50	–	24
miconic44	3110.23	–	–	34.02	–	28
miconic45	–	587.12	–	35.49	–	21
miconic46	–	–	–	166.68	–	27
miconic47	–	–	–	146.43	–	25
miconic48	–	2425.37	–	134.89	–	24
miconic49	3282.31	–	–	162.68	–	28
miconic50	–	–	–	149.78	–	26

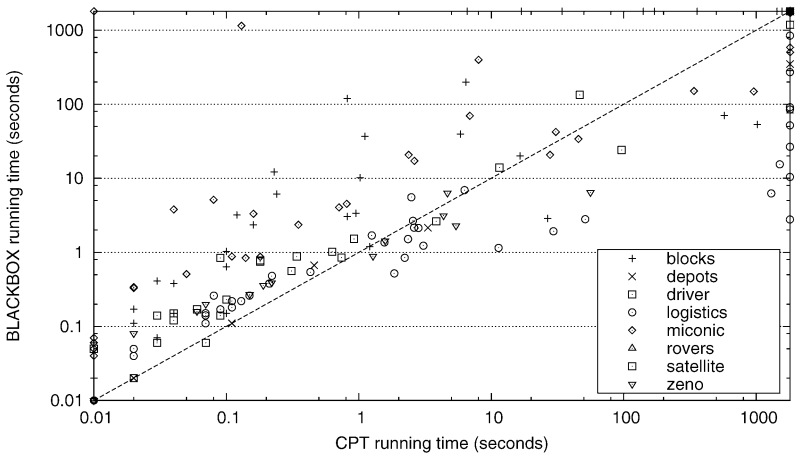

Fig. 1. Performance of CPT vs. BLACKBOX over parallel domains.

Table 5
Results for four parallel domains from IPC3

	CPU time (sec.)					Makespan
	CPT	BLACKBOX	SATPLAN	IPP	TP4	
depots01	0.02	0.02	0.14	0.00	0.08	5
depots02	0.11	0.11	0.34	0.01	1.45	8
depots03	0.46	0.67	68.68	0.20	143.38	12
depots04	3.32	2.14	353.33	0.38	–	14
depots05	–	349.22	–	–	–	20
depots06	–	–	–	–	–	–
driver01	0.02	0.02	0.15	0.00	0.08	6
driver02	0.03	0.14	1.10	0.02	2.03	9
driver03	0.03	0.06	0.20	0.01	0.15	7
driver04	0.04	0.12	0.38	0.07	4.54	7
driver05	0.06	0.17	0.44	0.75	69.01	8
driver06	0.07	0.06	0.17	0.01	2.37	5
driver07	0.09	0.14	0.21	0.08	32.79	6
driver08	0.10	0.23	0.32	1.92	168.21	7
driver09	0.34	0.88	32.67	5.73	584.05	10
driver10	0.31	0.56	7.02	9.65	2113.08	7
driver11	0.92	1.52	23.32	1.27	72.06	9
driver12	–	1186.49	–	–	–	16
satellite01	0.01	0.05	10.23	0.00	0.02	8
satellite02	0.09	0.84	265.85	0.01	12.38	12
satellite03	0.04	0.15	4.99	0.01	0.36	6
satellite04	0.18	0.78	129.91	4.10	–	10
satellite05	0.74	0.85	52.33	79.03	–	7
satellite06	0.18	0.75	25.25	40.96	–	8
satellite07	0.63	1.02	26.00	571.25	–	6
satellite08	46.59	133.79	295.17	–	–	8
satellite09	3.84	2.62	39.01	–	–	6
satellite10	96.44	24.10	193.68	–	–	8
satellite11	11.53	13.90	172.87	–	–	8
satellite12	–	–	–	–	–	–
zeno01	0.01	0.01	0.12	0.00	0.05	1
zeno02	0.02	0.08	0.18	0.00	0.06	5
zeno03	0.07	0.20	0.28	0.01	0.30	5
zeno04	0.06	0.16	0.22	0.00	0.63	5
zeno05	0.15	0.27	0.37	0.01	1.49	5
zeno06	0.19	0.36	0.91	0.02	40.72	5
zeno07	0.22	0.39	0.54	0.02	266.24	6
zeno08	1.28	0.89	5.50	0.12	2088.00	5
zeno09	1.58	1.44	45.62	0.38	–	6
zeno10	5.42	2.28	102.21	123.58	–	6
zeno11	4.36	3.13	140.87	17.55	–	6
zeno12	4.67	6.32	201.85	743.63	–	6
zeno13	56.03	6.44	353.82	–	–	7
zeno14	–	–	–	–	–	–

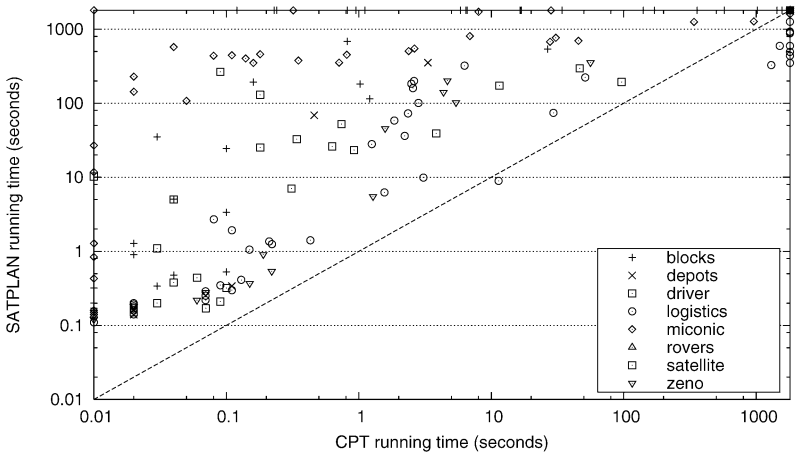

Fig. 2. Performance of CPT vs. SATPLAN04 over parallel domains.

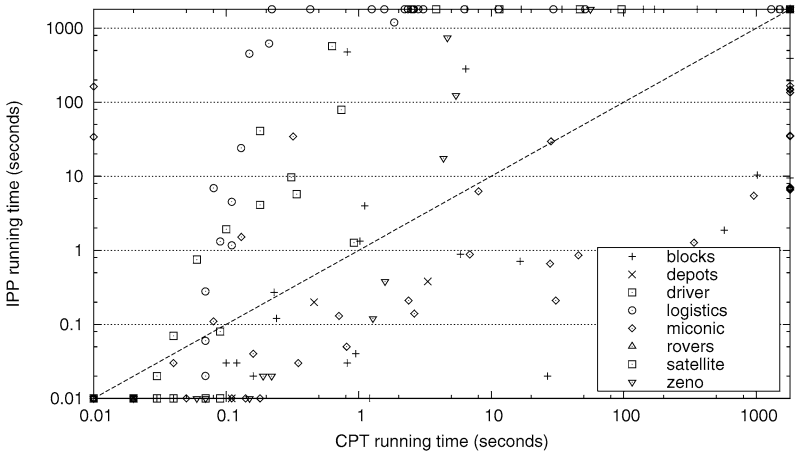

Fig. 3. Performance of CPT vs. IPP over parallel domains.

8. 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 key novelty in the planner and the source of its power, is the ability to represent and reason about supports, precedences, and causal links involving actions that are not in the plan. The experiments show that the resulting planner is faster than current optimal temporal planners and is competitive with the best parallel planners in the special case in which actions have all the same duration. The formulation extends the one in [45] that assumes that no ground action in the domain occurs more than once in the plan. This canonicity restriction is re-

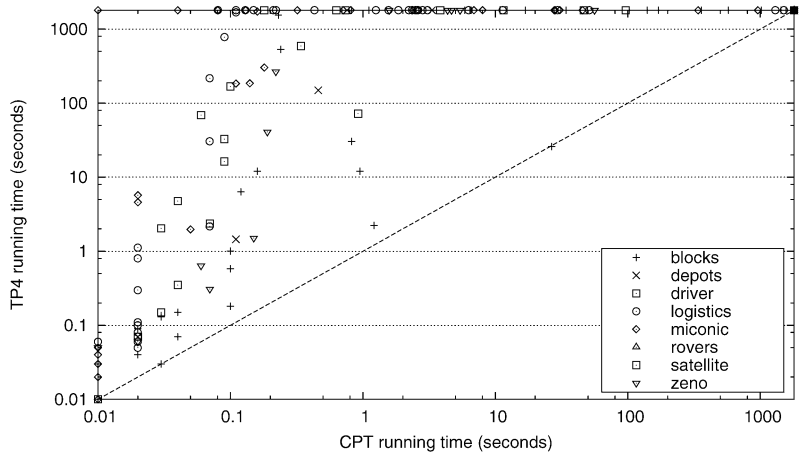

Fig. 4. Performance of CPT vs. TP4'04 over parallel domains.

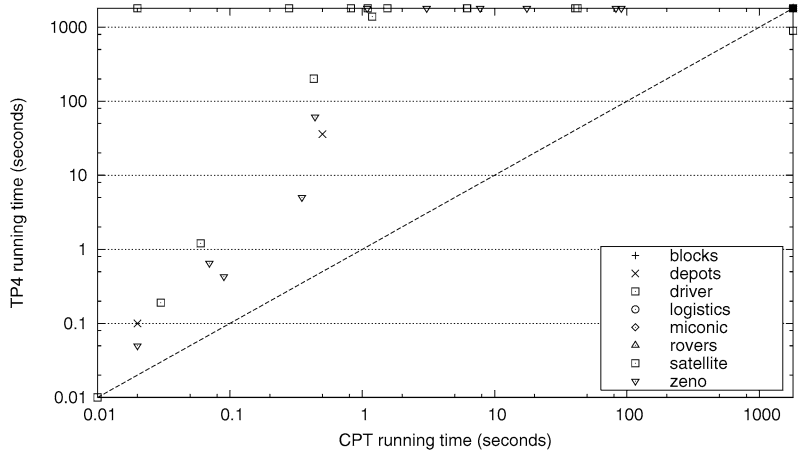

Fig. 5. Performance of CPT vs. TP4'04 over temporal domains.

moved by establishing a distinction between *action types* and *action tokens*, the latter being created dynamically during the search. The resulting scheme can be understood as providing a lazy implementation of an action domain with an infinite collection of action tokens or instances. Indeed, the action types are used as place holders for the information that is common to all the action instances of that type that have not yet made it into the plan. The move from canonical to general planning where ground actions can be repeated many times, involves however an overhead. In Tables 7–9, we actually compare the general CPT planner with the CPT planner with the canonicity restriction. The latter planner, that we refer to as CPT-CA in the tables, is a planner that is optimal only when some of the optimal plans are canonical. This happens automatically in domains like Blocks World for example, where all instances are canonical in this sense (they never require repeating the same ground

Table 6
Results for four temporal domains from IPC3

	CPU time (sec.)		Makespan		CPU time (sec.)		Makespan
	CPT	TP404			CPT	TP404	
driver01	0.02	4.18	91	depots01	0.02	0.08	28
driver02	–	365.89	92	depots02	0.50	19.73	36
driver03	0.03	0.18	40	depots03	–	–	–
driver04	–	–	–	depots04	–	–	–
driver05	40.67	–	51	depots05	–	–	–
driver06	–	–	–	depots06	–	–	–
driver07	0.43	45.52	40	zeno01	0.02	0.07	173
driver08	–	–	–	zeno02	0.07	0.28	592
driver09	–	–	–	zeno03	0.09	0.43	280
driver10	6.16	–	38	zeno04	1.09	–	522
driver11	–	–	–	zeno05	0.44	30.54	400
driver12	–	–	–	zeno06	0.35	4.87	323
satellite01	0.01	0.01	46	zeno07	3.07	–	665
satellite02	1.19	466.63	70	zeno08	17.52	–	522
satellite03	0.06	1.17	34	zeno09	90.64	–	522
satellite04	0.82	–	58	zeno10	82.62	–	453
satellite05	1.55	–	36	zeno11	7.77	–	423
satellite06	0.28	–	46	zeno12	–	–	–
satellite07	1.10	–	34	zeno13	–	–	–
satellite08	–	–	–	zeno14	–	–	–
satellite09	6.21	–	34				
satellite10	1897.84	–	43				
satellite11	42.32	–	46				
satellite12	–	–	–				

action twice). In general, however, when this assumption is not true, CPT-CA may result in non-optimal plans (non-optimality), or may even find no plan at all (incompleteness). Interestingly by looking at the tables, we only find four examples of non-optimality (`log-ipc09`, `log-ipc10`, `depots03` and `driver02`), and no example of incompleteness; indicating that while not valid, the canonicity restriction is often reasonable. At the same time, since the consideration of non-canonical plans involves an overhead, the canonical planner CPT-CA ends up actually solving more problems in the given time window (1 hour) than the general CPT planner. This is most prominent in the temporal DriverLog instances where the former solves 11 out of the 12 instances, while the latter solves only 5, but it is also true for Blocks World and Logistics. In addition, in all instances, with the four exceptions mentioned above, when both CPT and CPT-CA find a plan, CPT-CA finds a plan that is as good in less time. It remains an open challenge to determine the conditions under which restrictions like canonicity or suitable variations (e.g., that certain actions are 'canonical' but not others) can be detected and exploited. In the future, we would also like to analyze in further detail the constraints that are most critical in pruning the search space in CPT, and whether this pruning power can be further extended by explicating additional constraints in the formulation such as those encoding 'landmark' information [38].

Table 7
General planning in CPT vs. Restricted canonical planning in CPT over Blocks World and Logistics

Blocks World	CPU time (sec.)		Makespan		Logistics	CPU time (sec.)		Makespan	
	CPT	CPT-CA	CPT	CPT-CA		CPT	CPT-CA	CPT	CPT-CA
bw-12step	0.10	0.08	12	12	log.easy	0.02	0.02	9	9
bw-large.a	0.10	0.09	12	12	rocket.a	0.11	0.09	7	7
bw-large.b	1.02	0.98	18	18	rocket.b	0.08	0.06	7	7
bw-large.c	140.30	129.93	28	28	log.a	0.15	0.15	11	11
bw-large.d	–	–	–	–	log.b	1.85	0.38	13	13
bw-ipc01	0.00	0.00	6	6	log.c	2.22	0.53	13	13
bw-ipc02	0.01	0.01	10	10	log.d	2.82	1.33	14	14
bw-ipc03	0.01	0.01	6	6	log.d3	1.25	1.22	13	13
bw-ipc04	0.03	0.02	12	12	log.d1	–	121.12	–	17
bw-ipc05	0.01	0.01	10	10	log-ipc01	0.02	0.02	9	9
bw-ipc06	0.02	0.02	16	16	log-ipc02	0.02	0.02	9	9
bw-ipc07	0.04	0.03	12	12	log-ipc03	0.02	0.02	9	9
bw-ipc08	0.02	0.02	10	10	log-ipc04	0.02	0.02	9	9
bw-ipc09	0.03	0.03	20	20	log-ipc05	0.02	0.01	9	9
bw-ipc10	0.04	0.04	20	20	log-ipc06	0.01	0.01	3	3
bw-ipc11	26.63	3.04	22	22	log-ipc07	0.02	0.02	9	9
bw-ipc12	1.21	0.31	20	20	log-ipc08	0.02	0.02	9	9
bw-ipc13	0.16	0.10	18	18	log-ipc09	0.02	0.02	9	11
bw-ipc14	0.82	0.30	20	20	log-ipc10	0.09	0.09	12	13
bw-ipc15	0.10	0.09	16	16	log-ipc11	0.13	0.08	13	13
bw-ipc16	0.24	0.19	30	30	log-ipc12	0.07	0.06	11	11
bw-ipc17	0.95	0.49	28	28	log-ipc13	0.11	0.08	12	12
bw-ipc18	0.12	0.11	26	26	log-ipc14	0.07	0.06	11	11
bw-ipc19	0.23	0.22	34	34	log-ipc15	0.07	0.07	10	10
bw-ipc20	1018.47	88.93	32	32	log-ipc16	1.56	0.25	15	15
bw-ipc21	16.51	4.04	34	34	log-ipc17	0.21	0.18	12	12
bw-ipc22	–	1041.98	–	32	log-ipc18	0.43	0.23	13	13
bw-ipc23	–	2898.88	–	30	log-ipc19	3.06	0.28	15	15
bw-ipc24	1.11	0.56	34	34	log-ipc20	0.22	0.19	12	12
bw-ipc25	574.46	94.27	34	34	log-ipc21	11.39	0.46	15	15
bw-ipc26	5.86	1.82	34	34	log-ipc22	51.12	5.12	13	13
bw-ipc27	0.82	0.79	42	42	log-ipc23	2.60	1.10	13	13
bw-ipc28	6.43	1.60	44	44	log-ipc24	2.36	1.50	12	12
bw-ipc29	–	1672.96	–	38	log-ipc25	2.56	1.16	13	13
bw-ipc30	–	–	–	–	log-ipc26	2.50	1.16	13	13
bw-ipc31	1434.88	554.87	40	40	log-ipc27	29.54	7.57	12	12
bw-ipc32	6.57	2.91	52	52	log-ipc28	6.28	3.57	13	13
bw-ipc33	–	–	–	–	log-ipc29	–	–	–	–
bw-ipc34	1706.01	654.64	52	52	log-ipc30	1505.16	10.64	14	14
bw-ipc35	–	–	–	–	log-ipc31	–	–	–	–
bw-ipc36	–	–	–	–	log-ipc32	–	507.39	–	15
bw-ipc37	–	–	–	–	log-ipc33	1298.66	146.18	13	13
bw-ipc38	–	–	–	–	log-ipc34	–	34.80	–	15
bw-ipc39	34.15	8.93	62	62	log-ipc35	–	140.01	–	14
bw-ipc40	358.65	257.16	58	58	log-ipc36	–	–	–	–
bw-ipc41	–	–	–	–	log-ipc37	–	–	–	–
bw-ipc42	170.45	20.23	72	72	log-ipc38	–	–	–	–

(*continued on next page*)

Table 7 (*continued*)

Blocks World	CPU time (sec.)		Makespan		Logistics	CPU time (sec.)		Makespan	
	CPT	CPT-CA	CPT	CPT-CA		CPT	CPT-CA	CPT	CPT-CA
bw-ipc43	16.86	15.76	78	78	log-ipc39	–	–	–	–
bw-ipc44	1563.39	249.01	68	68	log-ipc40	–	–	–	–
bw-ipc45	–	–	–	–					
bw-ipc46	–	–	–	–					
bw-ipc47	–	–	–	–					
bw-ipc48	–	–	–	–					
bw-ipc49	–	–	–	–					
bw-ipc50	–	–	–	–					

Table 8
General planning in CPT vs. Restricted canonical planning in CPT over Miconic

	CPU time (sec.)		Makespan			CPU time (sec.)		Makespan	
	CPT	CPT-CA	CPT	CPT-CA		CPT	CPT-CA	CPT	CPT-CA
miconic01	0.00	0.00	4	4	miconic26	0.71	0.68	14	14
miconic02	0.00	0.00	3	3	miconic27	0.08	0.07	15	15
miconic03	0.00	0.00	4	4	miconic28	2.38	2.26	16	16
miconic04	0.00	0.00	4	4	miconic29	2.63	2.58	16	16
miconic05	0.00	0.00	4	4	miconic30	30.69	28.71	18	18
miconic06	0.00	0.00	6	6	miconic31	6.89	6.49	18	18
miconic07	0.00	0.00	6	6	miconic32	339.13	328.26	20	20
miconic08	0.00	0.01	6	6	miconic33	27.82	26.39	17	17
miconic09	0.00	0.00	6	6	miconic34	45.48	43.06	17	17
miconic10	0.00	0.00	6	6	miconic35	0.13	0.12	23	23
miconic11	0.01	0.01	8	8	miconic36	8.02	7.51	22	22
miconic12	0.01	0.01	10	10	miconic37	–	–	–	–
miconic13	0.01	0.01	8	8	miconic38	958.34	922.00	20	20
miconic14	0.01	0.01	9	9	miconic39	–	–	–	–
miconic15	0.01	0.01	8	8	miconic40	–	–	–	–
miconic16	0.02	0.01	12	12	miconic41	–	–	–	–
miconic17	0.02	0.02	11	11	miconic42	28.24	26.53	24	24
miconic18	0.11	0.10	14	14	miconic43	0.32	0.32	24	24
miconic19	0.14	0.14	14	14	miconic44	3110.23	3089.62	28	28
miconic20	0.18	0.17	14	14	miconic45	–	–	–	–
miconic21	0.35	0.32	14	14	miconic46	–	–	–	–
miconic22	0.81	0.76	15	15	miconic47	–	–	–	–
miconic23	0.05	0.04	10	10	miconic48	–	–	–	–
miconic24	0.16	0.14	14	14	miconic49	3282.31	3212.65	28	28
miconic25	0.04	0.04	16	16	miconic50	–	–	–	–

Acknowledgements

The first author thanks Gérard Verfaillie for comments on earlier versions of this paper and numerous discussions, and Patrick Haslum for his assistance on the use of TP4'04. Part of this work was done while the second author visited Nasa Ames and the Universita di Genova in the Summer of 2000. He thanks Nicola Muscettola and Enrico Giunchiglia for

Table 9
General planning in CPT vs. Restricted canonical planning in CPT over four parallel and temporal domains from IPC3

	Parallel domains				Temporal domains			
	CPU time (sec.)		Makespan		CPU time (sec.)		Makespan	
	CPT	CPT-CA	CPT	CPT-CA	CPT	CPT-CA	CPT	CPT-CA
depots01	0.02	0.02	5	5	0.02	0.02	28	28
depots02	0.11	0.10	8	8	0.50	0.17	36	36
depots03	0.46	0.79	12	13	–	–	–	–
depots04	3.32	1.43	14	14	–	18.73	–	40
depots05	–	–	–	–	–	–	–	–
depots06	–	–	–	–	–	–	–	–
driver01	0.02	0.02	6	6	0.02	0.01	91	91
driver02	0.03	0.07	9	10	–	355.68	–	92
driver03	0.03	0.03	7	7	0.03	0.03	40	40
driver04	0.04	0.03	7	7	–	29.28	–	52
driver05	0.06	0.05	8	8	40.67	0.52	51	51
driver06	0.07	0.07	5	5	–	46.33	–	52
driver07	0.09	0.08	6	6	0.43	0.22	40	40
driver08	0.10	0.10	7	7	–	2686.41	–	52
driver09	0.34	0.28	10	10	–	114.08	–	92
driver10	0.31	0.29	7	7	6.16	2.33	38	38
driver11	0.92	0.76	9	9	–	3365.36	–	65
driver12	–	–	–	–	–	–	–	–
satellite01	0.01	0.01	8	8	0.01	0.00	46	46
satellite02	0.09	0.06	12	12	1.19	0.56	70	70
satellite03	0.04	0.04	6	6	0.06	0.05	34	34
satellite04	0.18	0.15	10	10	0.82	0.61	58	58
satellite05	0.74	0.57	7	7	1.55	1.22	36	36
satellite06	0.18	0.18	8	8	0.28	0.26	46	46
satellite07	0.63	0.63	6	6	1.10	0.95	34	34
satellite08	46.59	41.82	8	8	–	1921.27	–	46
satellite09	3.84	3.87	6	6	6.21	5.58	34	34
satellite10	96.44	88.28	8	8	1897.84	1474.57	43	43
satellite11	11.53	10.91	8	8	42.32	30.96	46	46
satellite12	–	–	–	–	–	–	–	–
zeno01	0.01	0.02	1	1	0.02	0.02	173	173
zeno02	0.02	0.02	5	5	0.07	0.06	592	592
zeno03	0.07	0.06	5	5	0.09	0.09	280	280
zeno04	0.06	0.06	5	5	1.09	0.45	522	522
zeno05	0.15	0.15	5	5	0.44	0.40	400	400
zeno06	0.19	0.19	5	5	0.35	0.34	323	323
zeno07	0.22	0.22	6	6	3.07	1.42	665	665
zeno08	1.28	1.32	5	5	17.52	9.68	522	522
zeno09	1.58	1.66	6	6	90.64	38.18	522	522
zeno10	5.42	5.35	6	6	82.62	12.54	453	453
zeno11	4.36	4.36	6	6	7.77	7.81	423	423
zeno12	4.67	4.48	6	6	–	–	–	–
zeno13	56.03	50.22	7	7	–	–	–	–
zeno14	–	–	–	–	–	–	–	–

their hospitality and a number of useful discussions. He has also benefited from discussions with P. Haslum, P. Laborie, C. Beck, S. Kambhampati, D. Smith, A. Jonsson, J. Frank, and P. Morris. He also thanks Héctor Palacios for the related joint work in [36]. V. Vidal is partially supported by the "IUT de Lens", the CNRS and the "région Nord/Pas-de-Calais" under the COCOA program. H. Geffner is partially supported by Grant TIC2002-04470-C03-02, MCyT, Spain.

References

[1] F. Bacchus, The 2000 AI planning systems competition, Artificial Intelligence Magazine 22 (3) (2001) 47–56.
[2] P. Baptiste, C. Le Pape, W. Nuijten, Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems, Kluwer, Dordrecht, 2001.
[3] A. Blum, M. Furst, Fast planning through planning graph analysis, in: Proceedings of IJCAI-95, Montreal, Quebec, Morgan Kaufmann, San Mateo, CA, 1995, pp. 1636–1642.
[4] B. Bonet, H. Geffner, Planning as heuristic search, Artificial Intelligence 129 (1–2) (2001) 5–33.
[5] B. Bonet, G. Loerincs, H. Geffner, A robust and fast action selection mechanism for planning, in: Proceedings of AAAI-97, Providence, RI, MIT Press, Cambridge, MA, 1997, pp. 714–719.
[6] J. Carlier, E. Pinson, An algorithm for solving the job shop scheduling problem, Management Science 35 (2) (1989) 164–176.
[7] Y. Caseau, F.X. Josset, F. Laburthe, CLAIRE: Combining sets, search and rules to better express algorithms, in: Proceedings of ICLP-99, 1999, pp. 245–259.
[8] Y. Caseau, F. Laburthe, Improved CLP scheduling with task intervals, in: Proceedings of ICLP-94, MIT Press, Cambridge, MA, 1994, pp. 369–383.
[9] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence 49 (1991) 61–95.
[10] M.B. Do, S. Kambhampati, Solving planning-graph by compiling it into CSP, in: Proceedings of AIPS-00, 2000, pp. 82–91.
[11] S. Edelkamp, J. Hoffmann, The 4th international planning competition, at http://ipc04.icaps-conference.org, 2004.
[12] F. Focacci, A. Lodi, M. Milano, Solving TSPs with time windows with constraints, in: Proceedings of ICLP-99, MIT Press, Cambridge, MA, 1999, pp. 515–529.
[13] F. Focacci, M. Milano, Connections and integrations of dynamic programming and constraint programming, in: Proceedings of CP-AI-OR'01, 2001.
[14] M. Fox, D. Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains, J. Artificial Intelligence Res. (2003) 61–124.
[15] H. Geffner, Planning as branch and bound and its relation to constraint-based approaches, Technical report, Universidad Simón Bolívar, 2001, at www.tecn.upf.es/~hgeffner.
[16] P. Haslum, H. Geffner, Admissible heuristics for optimal planning, in: Proceedings of the Fifth International Conference on AI Planning Systems (AIPS-2000), 2000, pp. 70–82.
[17] P. Haslum, H. Geffner, Heuristic planning with time and resources, in: Proceedings of European Conference of Planning (ECP-01), 2001, pp. 121–132.
[18] A. Jonsson, P. Morris, N. Muscettola, K. Rajan, Planning in interplanetary space: Theory and practice, in: Proceedings of AIPS-2000, 2000, pp. 177–186.
[19] D. Joslin, M.E. Pollack, Is "early commitment" in plan generation ever a good idea?, in: Proceedings of AAAI-96, Portland, OR, 1996, pp. 1188–1193.
[20] S. Kambhampati, C. Knoblock, Q. Yang, Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning, Artificial Intelligence 76 (1–2) (1995) 167–238.
[21] S. Kambhampati, B. Srivastava, Universal classical planner: An algorithm for unifying state-space and plan-space planning, in: M. Ghallab, A. Milani (Eds.), New Directions in AI Planning, IOS Press, Amsterdam, 1996, pp. 61–78.

[22] H. Kautz, D. McAllester, B. Selman, Encoding plans in propositional logic, in: Proceedings of KR-96, 1996, pp. 374–384.
[23] H. Kautz, B. Selman, Unifying SAT-based and Graph-based planning, in: T. Dean (Ed.), Proceedings of IJCAI-99, Stockholm, Sweden, Morgan Kaufmann, San Mateo, CA, 1999, pp. 318–327.
[24] J. Koehler, B. Nebel, J. Hoffman, Y. Dimopoulos, Extending planning graphs to an ADL subset, in: S. Steel, R. Alami (Eds.), Recent Advances in AI Planning. Proceedings of 4th European Conf. on Planning (ECP-97), in: Lecture Notes in Artificial Intelligence, vol. 1348, Springer, Berlin, 1997, pp. 273–285.
[25] J. Koehler, K. Schuster, Elevator control as a planning problem, in: Proceedings of AIPS-00, 2000, pp. 331–338.
[26] P. Laborie, Algorithms for propagating resource constraints in AI planning and scheduling, Artificial Intelligence 143 (2003) 151–188.
[27] P. Laborie, M. Ghallab, Planning with sharable resources constraints, in: C. Mellish (Ed.), Proceedings of IJCAI-95, Montreal, Quebec, Morgan Kaufmann, San Mateo, CA, 1995, pp. 1643–1649.
[28] F. Laburthe, CHOCO: implementing a CP kernel, in: Proceedings of CP-00, in: Lecture Notes in Comput. Sci., vol. 1894, Springer, Berlin, 2000.
[29] O. Lhomme, Consistency techniques for numeric CSPs, in: Proceedings of IJCAI-93, Chambéry, France, Morgan Kaufmann, San Mateo, CA, 1993, pp. 232–238.
[30] D. Long, M. Fox, The 3rd international planning competition: Results and analysis, J. Artificial Intelligence Res. 20 (2003) 1–59.
[31] A. Mali, A. Kambhampati, On the utility of plan-space (causal) encodings, in: Proceedings of AAAI-99, Orlando, FL, 1999, pp. 557–563.
[32] D. McAllester, D. Rosenblitt, Systematic nonlinear planning, in: Proceedings of AAAI-91, Anaheim, CA, AAAI Press, Menlo Park, CA, 1991, pp. 634–639.
[33] D. McDermott, A heuristic estimator for means-ends analysis in planning, in: Proceedings of AIPS-96, 1996, pp. 142–149.
[34] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient SAT solver, in: Proceedings of DAC-01, 2001, pp. 530–535.
[35] X.L. Nguyen, S. Kambhampati, Reviving partial order planning, in: Proceedings of IJCAI-01, Seattle, WA, 2001, pp. 459–466.
[36] H. Palacios, H. Geffner, Planning as branch and bound: A constraint programming implementation, in: Proceedings of XXVIII Conf. Latinoamericana de Informática, 2002, pp. 239–251.
[37] J.S. Penberthy, D.S. Weld, Temporal planning with continuous change, in: Proceedings of AAAI-94, Seattle, WA, 1994, pp. 1010–1015.
[38] J. Porteous, L. Sebastia, J. Hoffmann, On the extraction, ordering, and usage of landmarks in planning, in: Proceedings of ECP-01, 2001, pp. 37–48.
[39] D. Smith, J. Frank, A. Jonsson, Bridging the gap between planning and scheduling, Knowledge Engrg. Rev. 15 (1) (2000) 61–94.
[40] D. Smith, D.S. Weld, Temporal planning with mutual exclusion reasoning, in: Proceedings of IJCAI-99, Stockholm, Sweden, Morgan Kaufmann, San Mateo, CA, 1999, pp. 326–337.
[41] S. Smith, C. Cheng, Slack-based heuristics for the constraint satisfaction scheduling, in: Proceedings of AAAI-93, Washington, DC, 1993, pp. 139–144.
[42] P. Van Beek, X. Chen, CPlan: a constraint programming approach to planning, in: Proceedings AAAI-99, Orlando, FL, 1999, pp. 585–590.
[43] P. Van Hentenryck, The OPL Optimization Programming Language, MIT Press, Cambridge, MA, 1999.
[44] P. Van Hentenryck, H. Simonis, M. Dincbas, Constraint satisfaction using constraint logic programming, Artificial Intelligence 58 (1–3) (1992) 113–159.
[45] V. Vidal, H. Geffner, Branching and pruning: An optimal temporal POCL planner based on constraint programming, in: Proceedings of AAAI-2004, San Jose, CA, 2004, pp. 570–577.
[46] D.S. Weld, An introduction to least commitment planning, AI Magazine 15 (4) (1994) 27–61.
[47] H.L.S. Younes, R.G. Simmons, VHPOP: Versatile heuristic partial order planner, J. Artificial Intelligence Res. 20 (2003) 405–430.
[48] Y. Zhang, R. Yap, Arc consistency on n-ary monotonic and linear constraints, in: Proceedings of CP 2000, 2000, pp. 470–483.