The New Version of CPT, an Optimal Temporal POCL Planner based on Constraint Programming

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

Sébastien Tabary
CRIL - Université d'Artois
rue de l'université - SP16
62307 Lens Cedex, FRANCE
tabary@cril.univ-artois.fr

Overview

CPT is a domain-independent temporal planner that combines a branching scheme based on Partial Order Causal Link (POCL) Planning with powerful and sound pruning rules implemented as constraints. Unlike other recent approaches that build on POCL planning (Nguyen & Kambhampati 2001; Younes & Simmons 2003), CPT is an optimal planner that minimizes makespan. The details of the planner and its underlying formulation are described in (Vidal & Geffner 2004; Vidal & Geffner 2006). CPT competed in the optimal track of IPC-4, where it got a second place.

The development of CPT is motivated by the limitation of heuristic state approaches to parallel and temporal planning that suffer from a high branching factor (Haslum & Geffner 2001) and thus have difficulties matching the performance of planners built on SAT techniques such as Blackbox (Kautz & Selman 1999). In CPT, all branching decisions (resolution of open supports, support threats, and mutex threats), generate binary splits, and nodes σ in the search correspond to 'partial plans' very much as in POCL planning.

While ideally, one would like to have informative lower bounds $f(\sigma)$ on the makespan $f^*(\sigma)$ of the best complete plans that expand σ, so that the partial plan σ can be pruned if $f(\sigma) \not\leq B$ for a given bound B, such lower bounds are not easy to come by in the POCL setting. CPT thus models the planning domain as a temporal constraint satisfaction problem, adds the constraint $f^*(\sigma) \leq B$ for a suitable bound B on the makespan, and performs limited form of constraint propagation in every node σ of the search tree. The novelty of CPT in relation to other temporal POCL planners such as IXTET (Laborie & Ghallab 1995) and RAX (Jonsson *et al.* 2000), that also rely on constraint propagation (and Dynamic CSP approaches such as (Joslin & Pollack 1996)), is the formulation that enables CPT to reason about actions a that are not yet in the plan. Often a lot can be inferred about such actions including restrictions about their possible starting times and supports. 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 (Blum & Furst 1995). CPT thus reasons with CSP variables that involve *all* the actions a in the domain and not only those present in the current plan, and for each such action, it deals with two 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. 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 $T(p, a) = T(a')$. A number of constraints enforce the correspondences among these variables. At the same time, the heuristic functions for estimating costs in a temporal setting, as introduced in (Haslum & Geffner 2001), are used to initialize variables domains and some 'distances' between actions (Van Beek & Chen 1999).

Currently, the semantics of the optimal temporal plans computed by CPT follows the one in (Smith & Weld 1999) where interfering actions (actions that delete a precondition or an effect of another one) are not allowed to overlap in time. This condition has been relaxed in PDDL 2.1 where interfering actions may overlap sometimes (e.g., when preconditions do not have to be preserved throughout the execution of the action). This restriction can in some domains produce slightly longer plans.

Additional pruning rules

CPT has been recently extended with several pruning rules. The primary goal of these rules was to give CPT the ability to solve planning problems in a suboptimal but backtrack-free way. Indeed, while fixing an upper bound on the makespan to a low value helps in pruning the search space, it has been remarked many times that fixing the bound to a high value (and thus, searching for a suboptimal plan) renders constraint-based planners particularly inefficient. To overcome this problem, we added many features to CPT, and we got interesting results in many classical benchmarks: BlocksWorld, Logistics, Gripper, Ferry, Satellite for example can be solved suboptimally without any backtrack. These results are reported in (Vidal & Geffner 2005). Some of these additions to CPT turn out to also help for optimal planning.

Impossible Supports

Many supports can be eliminated at preprocessing avoiding some dead-ends during the search. For example, the action $a' = putdown(b1)$ can never support the precondition $p = handempty$ of an action like $a = unstack(b1, b3)$. This is because action a has another precondition $p' = on(b1, b3)$

which is e-deleted[1] by a' (false after a') and which then would have to be reestablished by another action b before a. Yet it can be shown that in this domain, any such action b e-deletes p and is thus incompatible with the causal link $a'[p]a$.

More generally, let $dist(a', p, a)$ refer to a lower bound on the slack between actions a' and a in any valid plan in which a' *is a supporter of precondition* p *of* a. We show that for some cases, at preprocessing time, it can be shown that $dist(a', p, a) = \infty$, and hence, that a' can be safely removed from the domain of the variable $S(p, a)$ encoding the support of precondition p of a.

This actually happens when some precondition p' of a is not *reachable* from the initial situation that includes all the facts except those e-deleted by a' and where *the actions that either add or delete* p *are excluded.* The reason for this exclusion is that if a' supports the precondition p of a then it can be assumed that no action adding or deleting p can occur between a' and a (the first part is the systematicity requirement (McAllester & Rosenblitt 1991)). By a proposition being reachable we mean that it makes it into the so-called relaxed planning graph; the planning graph with the delete lists excluded (Hoffmann & Nebel 2001).

This simple test prunes the action $putdown(b_1)$ as a possible support of the precondition $handempty$ of action $unstack(b_1, b_3)$, the action $stack(b_1, b_3)$ as a possible support of precondition $clear(b_1)$ of $pickup(b_1)$, etc.

Unique Supports

We say that an action *consumes* an atom p when it requires and deletes p. For example, the actions $unstack(b_3, b_1)$ and $pickup(b_2)$ both consume the atom $handempty$. In such cases, if the actions make it into the plan, it can be shown that their common precondition p must have different supports. Indeed, if an action a deletes a precondition of a', and a' deletes a precondition of a, a and a' are incompatible and cannot overlap in time according to the semantics. Then either a must precede a' or a' must precede a, and in either case, the precondition p needs to be established at least twice: one for the first action, and one for the second. The constraint $S(p, a) \neq S(p, a')$ for pairs of actions a and a' that consume p, ensures this, and when one of the support variables $S(p, a)$ or $S(p, a')$ is instantiated to a value b, b is immediately removed from the domain of the other variable.

Distance Boosting

The distances $dist(a, a')$ precomputed for all pairs of actions a and a' provide a lower bound on the slack between the end of a and the beginning of a'. In some cases, this lower bound can be easily improved, leading to stronger inferences. For example, the distance between the actions $putdown(b_1)$ and $pickup(b_1)$ is 0, as it is actually possible to do one action after the other. Yet the action $putdown(b_1)$ followed by $pickup(b_1)$ makes sense only if some other action using the effects of the first, occurs between these two, as when block b_1 is on block b_2 but needs to be moved on top of the block beneath b_2.

Let us say that an action a *cancels* an action a' when 1) every atom added by a' is e-deleted by a, and 2) every atom added by a is a precondition of a'. Thus, when a cancels a', the sequence a', a does not add anything that was not already true before a'. For example, $pickup(b_1)$ cancels the action $putdown(b_1)$.

When an action a cancels a', and there is a precondition p of a that is made true by a' (i.e., p is added by a' and is mutex with some precondition of a'), the distance $dist(a', p, a)$ introduced above becomes ∞ if all the actions that use an effect of a' e-delete p. In such case, as before, the action a' can be excluded from the domain of the $S(p, a)$ variable. Otherwise, the distance $dist(a', a)$ can be increased to $\min_b[dist(a', b) + dist(b, a)]$ with b ranging over the actions different than a and a' that either use an effect of a' but do not e-delete p or do not use necessarily an effect of a' but add p (because a' may be followed by an action c before a that e-deletes p but only if there is another action b between c and a that re-establishes p).

In this way, the distance between the actions $putdown(a)$ and $pickup(a)$ in Blocks is increased by 2, the distance between $sail(a, b)$ and $sail(b, a)$ in Ferry is increased by 1, etc. The net effect is similar to pruning cycles of size two in standard heuristic search. Pruning cycles of larger sizes, however, appears to be more difficult in the POCL setting, although similar ideas can potentially be used for pruning certain sequences of commutative actions.

Improvement of the search algorithm

The original version of CPT performs a very basic backtracking search: its relative efficiency mainly comes from the look-ahead techniques encoded into the pruning rules, and from heuristics adapted to temporal planning. But even if an advanced look-ahead technique is used, one can be interested by looking for the reason of an encountered dead-end as finding the ideal ordering of variables is intractable in practice. A dead-end corresponds to a conflict between a subset of decisions (variable assignments) performed so far. In fact, it is relevant to prevent thrashing[2] to identify the most recent decision (let us call it the culprit one) that participates to the conflict. Indeed, once the culprit has been identified, we know that it is possible to safely backtrack up to it – this is the role of look-back techniques such as CBJ (Conflict-directed BackJumping) (Prosser 1993) and DBT (Dynamic Backtracking) (Ginsberg 1993).

A new approach as been recently proposed in (Lecoutre *et al.* 2006) to (indirectly) backtrack to the culprit of the last encountered dead-end. To achieve it, the leaf conflict variable becomes in priority the next variable to be selected as long as the successive assignments that involves it render

[1] An action a is said to *e-delete* 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, if a e-deletes p, p is false after doing a; see (Nguyen & Kambhampati 2001).

[2] Thrashing is the fact of repeatedly exploring the same subtrees. This phenomenon deserves to be carefully studied as an algorithm subject to thrashing can be very inefficient.

the network arc inconsistent. It then corresponds to checking the singleton consistency of this variable from the leaf towards the root of the search tree until a singleton value is found. In other words, the variable ordering heuristic is violated, until a backtrack to the culprit variable occurs and a singleton value is found. It is important to remark that, contrary to sophisticated backjump techniques, this technique can be grafted in a very simple way to a tree search algorithm without any additional data structure. This has been implemented very easily into CPT, making it able to solve difficult problems that were previously out of reach.

A note about the implementation

The first version of CPT planner was implemented using the Choco CP library (Laburthe 2000) that operates on top of Claire (Caseau, Josset, & Laburthe 1999), a high-level programming language that compiles into C++. Due to a number of restrictions of this language, we made a completely new implementation using the C language. This implementation is based on a minimal Constraint Programming engine inspired by the Choco library, offering all the basic needs: CP variables with enumerated and bounded domains, automatic propagation on the change of the domains based on events (instantiation, removal, lower bound increased, ...), a complete backtrack mechanism for undoing the changes, and a basic backtracking algorithm. The constraints of CPT are implemented with propagation rules which are triggered by the underlying CP engine. This implementation is by far more efficient than the original one, also having very minimal memory requirements (except in some benchmark domains such as ZenoTravel, where the formulation of the domain by itself leads to a high number of variables with very large domains). This new version will soon be available on the CPT web page[3]. We also plan to release the minimal CP engine of CPT as a separate package, as it is completely independent of CPT and could serve as a basis for many different applications. As an example, CPT has been recently used in an evolutionary based approach of multi-objective temporal planning (Schoenauer, Savéant, & Vidal 2006).

Acknowledgments

Most of the work on CPT has been made with Héctor Geffner, as well as some material of this abstract borrowed from our common papers. Many thanks to him.

References

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

[Caseau, Josset, & Laburthe 1999] Caseau, Y.; Josset, F. X.; and Laburthe, F. 1999. CLAIRE: Combining sets, search and rules to better express algorithms. In *Proceedings of ICLP-99*, 245–259.

[Ginsberg 1993] Ginsberg, M. 1993. Dynamic backtracking. *Artificial Intelligence* 1:25–46.

[Haslum & Geffner 2001] Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proceedings of European Conference of Planning (ECP-01)*, 121–132.

[3] http://www.cril.univ-artois.fr/∼vidal/index.en.html

[Hoffmann & Nebel 2001] Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 2001:253–302.

[Jonsson *et al.* 2000] Jonsson, A.; Morris, P.; Muscettola, N.; and Rajan, K. 2000. Planning in interplanetary space: Theory and practice. In *Proceedings of AIPS-2000*, 177–186.

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

[Kautz & Selman 1999] Kautz, H., and Selman, B. 1999. Unifying SAT-based and Graph-based planning. In Dean, T., ed., *Proceedings of IJCAI-99*, 318–327. Morgan Kaufmann.

[Laborie & Ghallab 1995] Laborie, P., and Ghallab, M. 1995. Planning with sharable resources constraints. In Mellish, C., ed., *Proceedings of IJCAI-95*, 1643–1649. Morgan Kaufmann.

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

[Lecoutre *et al.* 2006] Lecoutre, C.; Sais, L.; Tabary, S.; and Vidal, V. 2006. Last conflict based reasoning. In *Proceedings of ECAI-2006* (to appear).

[McAllester & Rosenblitt 1991] McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of AAAI-91*, 634–639. Anaheim, CA: AAAI Press.

[Nguyen & Kambhampati 2001] Nguyen, X. L., and Kambhampati, S. 2001. Reviving partial order planning. In *Proceedings of IJCAI-01*, 459–466.

[Prosser 1993] Prosser, P. 1993. Hybrid algorithms for the constraint satisfaction problems. *Computational Intelligence* 9(3):268–299.

[Schoenauer, Savéant, & Vidal 2006] Schoenauer, M.; Savéant, P.; and Vidal, V. 2006. Divide-and-evolve: a new memetic scheme for domain-independent temporal planning. In *Proceedings of EvoCOP-2006*, 247–260.

[Smith & Weld 1999] Smith, D., and Weld, D. S. 1999. Temporal planning with mutual exclusion reasoning. In *Proceedings of IJCAI-99*, 326–337.

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

[Vidal & Geffner 2004] Vidal, V., and Geffner, H. 2004. Branching and pruning: An optimal temporal POCL planner based on constraint programming. In *Proceedings of AAAI-2004*, 570–577.

[Vidal & Geffner 2005] Vidal, V., and Geffner, H. 2005. Solving simple planning problems with more inference and no search. In *Proceedings of CP-2005*, 682–696.

[Vidal & Geffner 2006] Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence* 170(3):298–335.

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