Solving Simple Planning Problems with More Inference and No Search

Vincent Vidal and Héctor Geffner

Abstract

Many problems used in AI planning including Blocks, Logistics, Gripper, Satellite, and others lack the interactions that characterize puzzles and can be solved non-optimally in low polynomial time. They are indeed easy problems for people, although as with many other problems in AI, not always easy for machines. In this work, we study the type of inferences that are required in a domain-independent planner for solving simple problems such as these in a backtrack-free manner by performing polynomial node operations. For this, we make use of the optimal temporal planner CPT which combines a POCL branching scheme with strong inference mechanisms, and show that a few simple and general additional inference mechanisms suffice to render the search over various domains backtrack free. This is an interesting empirical finding, we believe, that may contribute to the development of more robust automated planners, and to a better understanding of human planning. Significant performance gains in relation to CPT are also reported.

Introduction

Many problems used in AI planning including Blocks, Logistics, and others lack the interactions that characterize puzzles and can be solved non-optimally in low polynomial time. They are indeed easy problems for people, although as with many other problems in AI, not always easy for machines. In this work, we want to study the type of inferences that are required in a domain-independent planner for solving simple problems such as these in a backtrack-free manner by performing polynomial operations in every node. For this, we make use of the optimal temporal planner CPT which combines a POCL branching scheme with strong inference mechanisms (Vidal and Geffner 2004a), and show that a few simple and general additional inference mechanisms suffice to render the search over various domains backtrack free.

We will refer to domain-independent planners that aim to solve simple problems in a backtrack-free manner by performing low polynomial operations in every node, as *easy planners*. The development of easy planners, we believe, is a crisp and meaningful goal which may contribute not only to the development of more robust automated planners, but also to a better understanding of human planning. Humans indeed are quite good at solving these simple problems, and while it is often assumed that this ability is the result of domain-dependent strategies, our results suggest that they may also result from simple but general inference mechanisms.

Easy planners are suboptimal planners, but while suboptimal planners aim to solve problems, and optimal planners aim to solve problems optimally, easy planners aim to solve simple problems with low polynomial computations per node and *no search.* By itself, CPT like other SAT and constraint-based optimal planners (Kautz and Selman 1999; Rintanen 1998; Do and Kambhampati 2000), does not make for a good suboptimal planner and much less for an easy planner. Indeed, while SAT and constraint-based planners can be used with large, non-optimal planning horizons, they face two problems:

1. SAT and CSP encodings based on one variable per time point, as normally used, become too large to handle for large planning horizons,
2. the constraint that requires the goals to be true at the planning horizon becomes ineffective when the horizon is set too high.

In CPT the first is not a problem because, being a temporal planner, CPT uses temporal rather than boolean encodings. Thus, the use of a large bound on the admissible makespan of plans has a direct effect on the *domain* of the temporal variables but not in their *number*.

CPT, on the other hand, does not escape from Problem 2: with a large bound on the makespan, the search becomes less constrained and focused, and even problems that are solved backtrack free with the optimal bound are not solved at all after thousands of backtracks when a larger bound is used instead. In this work, we tackle this problem by extending the inferential capabilities of CPT so that it relies less on inferences drawn from the bounding constraint and more on domain-independent inferences not captured by CPT.

The paper is organized as follows. We first review the CPT planner, discuss its strength as an optimal planner and its weakness as a suboptimal planner, and introduce extensions of the inferential machinery of CPT that render a search backtrack free over wide range of domains. We then evaluate the resulting planner, eCPT, and discuss implications and open ends.

CPT

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 (Vidal and Geffner 2004a). The key novelty in CPT in relation to other formulations (McAllester and Rosenblitt 1991; Kambhampati *et al.* 1995; Weld 1994) is the ability to reason about supports, precedences, and causal links involving actions that are not in the plan. In this way, CPT can prune the start time and supports of actions that are not yet in the plan, rule out actions from the plan, detect failures early on, etc.

CPT uses a simple extension of the Strips language that accommodates concurrent actions with integer durations. A temporal planning problem is a tuple $P = \langle A, I, O, G \rangle$ where A is a set of ground atoms, $I \subseteq A$ and $G \subseteq A$ represent the initial and goal situations, and O is the set of ground Strips operators, each with precondition, add, and delete list $pre(a)$, $add(a)$, and $del(a)$, and *duration* $dur(a)$. As is common in POCL Planning, there are also the 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 (Blum and Furst 1995), two actions a and a' interfere when one deletes a precondition or positive effect of the other. CPT follows the simple model of time in (Smith and Weld 1999) where interfering actions cannot overlap in time, and produces valid plans with minimum *makespan.*

The basic formulation of the CPT planner can be described in four parts: *preprocessing, variables, constraints, and branching.* For simplicity, we follow (Vidal and Geffner 2004a) and assume that no action in the domain can be done more than once in the plan. This restriction is removed in the last version of CPT, which is the one that we use, that introduces a distinction between action types and tokens. Such details, however, reported in (Vidal and Geffner 2004b), are not needed here and are omitted.

Preprocessing

In the preprocessing phase, CPT computes the heuristic values $h^2_T(a)$ and $h^2_T(\{p, q\})$ for each action $a \in O$ and each atom pair $\{p, q\}$ as in (Haslum and Geffner 2001). The values provide lower bounds on the times to achieve the preconditions of a and the pair of atoms p, q, from the initial situation I. The *(structural) mutexes* are then identified as the pairs of atoms p, q for which $h^2_T(\{p, q\}) = \infty$. 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; (Nguyen and Kambhampati 2001).

In addition, the simpler heuristic h^1_T is used for defining *distances* between actions (Van Beek and Chen 1999). For each action $a \in O$, the h^1_T heuristic is computed from an initial situation I_a that includes all facts *except those that are e-deleted by* a. The distances $dist(a, a')$ are then set to the resulting $h^1_T(a')$ values. 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. They are not symmetric in general and their calculation, which remains polynomial, involves the computation of the h^1_T heuristic $|O|$ times.

Variables and Domains

The state 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 plan; $InPlan(Start) = InPlan(End) = 1$ (true)

Variables $T(a)$, $S(p, a)$, and $T(p, a)$ associated with actions a which are not either in or out of the current plan (i.e., actions for which the $InPlan(a)$ variable is not set to either 0 or 1 yet) 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 in (Vidal and Geffner 2004a).

Constraints

The constraints correspond basically to disjunctions, rules, and precedences, or their combination. Temporal constraints are propagated by bounds consistency (Marriot and Stuckey 1999). The constraints apply to all actions $a \in O$ and all $p \in pre(a)$; we use $\delta(a, a')$ to stand for $dur(a) + dist(a, a')$.

- **Bounds:** For all $a \in O$, $T(Start) + dist(Start, a) \leq T(a)$ and $T(a) + dist(a, End) \leq T(End)$
- **Preconditions:** Supporter a' of precondition p of a must precede a by an amount that depends on $\delta(a', a)$:
$$T(a) \ \geq \min_{a' \in [D(S(p,a))]} (T(a') + \delta(a', a))$$
$$T(a') + \delta(a', a) > T(a) \rightarrow S(p, a) \neq a'$$
- **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
$$T(a') + dur(a') + \min_{a'' \in D[S(p,a)]} dist(a', a'') \leq T(p, a)$$
$$\vee \ T(a) + \delta(a, a') \leq T(a')$$

- **Mutex Constraints:** For effect-interfering a and a'[1]

$$T(a) + \delta(a, a') \leq T(a') \ \vee \ T(a') + \delta(a', a) \leq T(a)$$

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

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

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

$$\min_{a' \in D[S(p,a)]} T(a') \ \leq \ T(p, a) \ \leq \ \max_{a' \in D[S(p,a)]} T(a')$$

Branching

As in POCL planning, branching in CPT proceeds by iteratively selecting and fixing flaws in non-terminal states σ, 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* $T(End) = B$ where B is the current bound on the makespan, which in the optimal setting is set to a lower bound, and is then increased until a plan is found. 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 in POCL planning expressed in terms of the temporal and support variables, with the addition of 'mutex threats':

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

Flaws are selected for repair in the following order: first Support Threats (ST's), then Open Conditions (OC's), and finally Mutex Threats (MT's). ST's and MT's are repaired by posting precedence constraints, while OC's are repaired by choosing a supporter, as usual in POCL planning.

eCPT

CPT is an optimal temporal planner with good performance which is competitive with the best SAT parallel planners when actions have uniform durations. At the same time, for non-optimal planning, CPT has the advantage that the size of the encodings does not grow with the bound; indeed the bound in CPT enters only through the constraint $T(End) = B$, which affects the domains of the variables but not their number. In spite of this, however, CPT does not make for a good suboptimal planner, because like SAT and CSP planners it still relies heavily on the bounding constraint which becomes ineffective for large values of B.

[1]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.

Figure 1 shows the performance of CPT for the Tower-n problem for several values of n and several horizons B. Tower-n is the problem of assembling a specific tower of n blocks which are initially on the table. This is a trivial problem for people, but as shown in (Vidal and Geffner 2004a), it is not trivial for most optimal planners. CPT, however, solves this problem optimally *backtrack free* for any value of n. As the figure shows, however, the times and the number of backtracks increase when the horizon B is increased above the optimal bound, and for large values of B, CPT cannot solve these problems after thousands of seconds and backtracks.

The figure also shows the performance of eCPT, the planner that it can be seen, while the performance of CPT degrades with the increase of the bound B, the performance of eCPT remains stable, and actually *backtrack free* for the different values of B. eCPT exploits the flexibility afforded by the Constraint-Programming formulation underlying CPT, extending it with inferences that do not rely as much on the bound, and which produce a backtrack-free behavior across a wide range of simple domains. In this section we focus on such inferences.

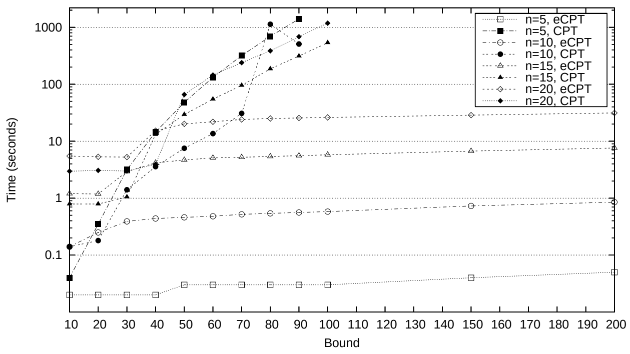

Figure 1: Performance of CPT and eCPT over Tower-n for various numbers of blocks n and bounds B. Curves that diverge correspond to CPT; curves that remain stable to eCPT.

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 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 and 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 and 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 can 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.

Qualitative Precedences

Unlike traditional POCL planners, CPT reasons with *temporal precedences* of the form $T(a) + \delta(a, a') \leq T(a')$ rather than with *qualitative precedences.* CPT is a CP-based temporal planner and such a choice arises naturally from the representation used. Yet, the constraint propagation mechanism, bounds consistency, is incomplete, and in a planning context, it is often too weak. In particular, bounds consistency does not capture *transitivity:* namely from the constraints $A < B$ and $B < C$, it does not entail $A < C$. Indeed if the initial domains of the variables A, B, and C is $[1, \ldots, 100]$, bounds consistency reduces the domains to $[1, \ldots, 98]$, $[2, \ldots, 99]$, and $[3, \ldots, 100]$ respectively, which do not make $A < C$ true for all value combinations. Transitivities, however, are important in planning, and thus eCPT incorporates, in addition to temporal precedences, qualitative precedences of the form $a \prec a'$ *not limited to the actions* a *and* a' *in the plan.* Such qualitative precedences are obtained every time a temporal precedence is asserted or entailed, and are kept closed under transitivity.[2] When a new qualitative precedence $a \prec a'$ is found, the transitive closure is computed as follows: if a belongs to the current partial plan, then for all a'' such that $a'' \prec a$, $a'' \prec a'$ is recorded; and if a' belongs to the plan, then for all a'' such that $a' \prec a''$, $a \prec a''$ is recorded. The same updates are incrementally performed for an existing relation $a \prec a'$ with a or a' not in the plan, as soon as a or a' make it into the plan.

Then two inference rules make use of these qualitative precedences for pruning further the domains of the support variables:

- for an action a' in the plan that adds a precondition p of an action a: if $a \prec a'$ then $S(p, a) \neq a'$
- for an action a' that adds a precondition p of an action a and an action b in the plan that e-deletes p: if $a' \prec b$ and $b \prec a$, then $S(p, a) \neq a'$

Action Landmarks

Like all POCL planners, CPT starts with a partial plan with two actions only: $Start$ and End. In many cases, however, it

[2]Temporal precedences are asserted as a result of the branching decisions corresponding to support and mutex threats, and are inferred when either supports are asserted or inferred, or when one of the disjuncts in a causal link or mutex constraint becomes false.

is possible to infer easily that certain other actions must be in the plan as well. For example, if a block b_1 must be moved but is beneath two blocks b_3 and b_2 in that order, then the actions $unstack(b_3, b_2)$ and $unstack(b_2, b_1)$ will have to be taken at some point, and moreover, the first must precede the second. In eCPT we identify such necessary actions and a partial order on them in a preprocessing step, following the idea of *landmarks* introduced in (Porteous *et al.* 2001), in the form presented in (Zhu and Givan 2004). An action a is a *landmark* if the action End is not *reachable* when the action a is excluded from the domain (as mentioned above, an action a is reachable when it makes it into the relaxed planning graph). Also, a landmark action a *precedes* a landmark action b, when b is not reachable when the action a is excluded. Action landmarks and the partial order on them are computed in the preprocessing step and are included in the initial state of the planner along with the actions $Start$ and End. This involves the calculation of $|O|$ relaxed planning graphs, one for each action in the domain.

Branching and Heuristics

eCPT retains the same branching scheme as CPT and the same ordering: it first branches on support threats (ST's), then on open conditions (OC's), and finally on mutex threats (MT's). The heuristic for selecting the support threats and open conditions however, are slightly different, although we have found that eCPT, working with large bounds, is much less sensitive than CPT to the various heuristics. Often, the difference is that some heuristics yield few backtracks in some of the domains, while others yield none.

Support threats $\langle a', S(p,a)\rangle$ are selected in eCPT minimizing $T_{min}(a)$, breaking ties by first minimizing $T_{max}(p,a)$, and then with the slack based criterion used in CPT. Open supports $S(p,a)$ are selected minimizing $T_{max}(p,a)$, breaking ties minimizing $slack(a',a)$ where a' is the producer of a in $D[S(p,a)]$ with min $T_{min}(a')$. Also the constraint posted in the second case is $S(p,a) = a'$, and if that fails, $S(p,a) \neq a'$.

Experimental Results

We report results for assessing eCPT along three dimensions: as an 'easy planner' able to solve problems backtrack free, as a suboptimal planner in comparison to state-of-the-art planners like FF (Hoffmann and Nebel 2001), and as an optimal temporal planner in comparison with CPT. The instances and domains are all from the 2nd and 3rd Int. Planning Competitions (Bacchus 2001; Fox and Long 2003), and the results have been obtained using a Pentium IV machine running at 2.8Ghz, with 1Gb of RAM, under Linux. The time limit for each problem is 30 minutes, and all times include preprocessing. The bound B on the makespan for suboptimal eCPT is fixed to 200, and all actions are assumed to have unit duration.

Table 1 shows for each domain, the total number of instances, the number of instances solved by eCPT, the number of instances solved backtrack free (and in parenthesis, the max number of backtracks over problems solved with backtracks), and the max number of nodes generated (in POCL planning, this number is different than the number of actions in the plan). For illustration purposes, the number of instances solved and the corresponding max number of nodes generated are reported also for FF. As it can be seen, eCPT solves 339 out of 350 instances, 336 of them backtrack free, including all the instances of Blocks, Ferry, Logistics, Gripper, Miconic, Rovers and Satellite (the 11 unsolved instances are actually all caused by memory limitations in the Claire language rather than time). This is quite remarkable, we think; these are instances that were challenging until very recently. eCPT solves actually 3 instances more than FF over this set of problems, eCPT having best relative coverage in Blocks and DriverLog, and FF in Depots and Zeno.

In the last domain from IPC-3, Freecell, FF solves more instances than eCPT, which no longer exhibits a backtrack-free behavior. This domain, however, causes difficulties to FF as well due to the presence of dead-ends (Hoffmann 2001).

		eCPT			FF	
	#pbs	solved	b.-free (max b.)	max nd	solved	max nd
blocks	50	50	50(0)	275	42	146624
depots	20	18	16(4)	285	19	166141
driver	20	17	16(5)	176	15	4657
ferry	50	50	50(0)	1176	50	201
gripper	50	50	50(0)	201	50	200
logistics	50	50	50(0)	273	50	2088
miconic	50	50	50(0)	131	50	76
rovers	20	20	20(0)	207	20	3072
satellite	20	20	20(0)	249	20	5889
zeno	20	14	14(0)	70	20	933

Table 1: eCPT vs. FF: Coverage over various 'simple' domains, showing # problems solved, backtrack free (max # backtracks), and max # of nodes generated

Table 2 provides further details on a few instances. As it can be seen, the runtimes for eCPT tend to scale well although they do not compete with the runtimes of FF (except a few instances in Depots), even if FF often generates orders-of-magnitude many more nodes. Plan quality measured in the number of actions in the plan is better for FF in domains like Logistics or DriverLog, which may have to do with the fact that eCPT computes concurrent plans.

Finally, Figure 2 compares eCPT vs. CPT as *optimal* planners over all the instances. eCPT solves more instances than CPT generating many fewer nodes and often running orders-of-magnitude faster (although not always so as the additional overhead sometimes does not pay off). As *suboptimal planners*, eCPT solves 339 out of the 350 instances, while CPT solves only 65 instances, with runtimes often above 1000 seconds, resulting usually in poor plans.

Discussion

We think that the task of solving simple planning problems in a domain-independent way with no search, by performing low polynomial operations in every node, is a crisp and meaningful goal, which may contribute to the development of more robust planners, and to a better understanding of human planning. In this work we have shown that this goal can be achieved in the temporal planner CPT, over a wide

	CPU time (sec.)		Actions		Nodes	
	eCPT	FF	eCPT	FF	eCPT (bkts)	FF
bw-ipc48	59.51	-	74	-	281 (0)	-
bw-ipc49	78.37	-	80	-	282 (0)	-
bw-ipc50	85.09	0.02	88	86	235 (0)	195
log-ipc48	50.56	0.20	164	142	261 (0)	515
log-ipc49	51.54	0.50	176	171	273 (0)	1252
log-ipc50	50.39	0.43	161	154	245 (0)	1147
depots06	66.23	-	68	-	160 (0)	-
depots07	1.27	0.01	28	25	68 (0)	142
depots08	13.13	579.89	75	43	206 (0)	172478
driver14	5.40	0.09	48	45	75 (0)	1209
driver15	39.91	0.03	69	44	130 (0)	161
driver16	147.15	-	107	-	163 (5)	-

Table 2: eCPT vs. FF: further details on a few instances

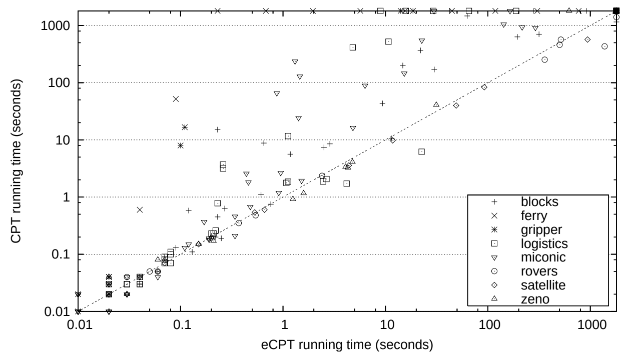

Figure 2: eCPT vs. CPT for optimal planning. Area above diagonal is where eCPT is faster, area below diagonal is where CPT is faster. Problems on the top border are unsolved by CPT, on the right border are unsolved by eCPT.

range of domains, by the addition of a few simple and general inference mechanisms. The inferences are tuned up for rendering the search backtrack free rather than fast, and have been obtained from observing the behavior of CPT over various domains, and accounting for inferences that were missing and thought to be responsible of unnecessary backtracks. The fact that this fine grain analysis is possible, and that the results can be easily incorporated into the planner, is a clear benefit of the CP formulation, which thus provides a way for making use of (human) *domain-specific* analysis for improving the performance of a *domain-independent* planner. We have also empirically evaluated the resulting planner eCPT, as a suboptimal and optimal planner, and have shown significant gains over CPT.

The finding that a few *inference rules* is all that it takes to render the search backtrack free in domains which until recently were considered challenging for planners, bears some similarity with the *empirical observation* in (Bonet *et al.* 1997) that a simple domain-independent *heuristic function* can effectively guide the search for plans in many domains, an idea exploited in many current planners. The two devices for taming the search, however, are different: heuristic estimators provide *numeric* information to weight alternatives, the inference rules provide *structural* information to discard alternatives. We believe that it should be possible to *prove* some domains backtrack free for eCPT, and in this way identify new abstract classes of tractable problems. Current classes, as defined in (Bäckström and Nebel 1995; Brafman and Domshlak 2003), remain somewhat narrow, and do not account for the tractability of existing benchmarks (Helmert 2003). In the future, we want to investigate minimal sets of rules for rendering the search backtrack free over the domains studied, simple inferences that are currently not accounted for and which may be causing backtracks in some domains, and ways for making the implementation more efficient.

References

F. Bacchus. The 2000 AI Planning Systems Competition. *AI Magazine*, 22(3):47–56, 2001.

C. Bäckström and B. Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995.

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

B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proc. AAAI-97*, pages 714–719, 1997.

R. Brafman and C. Domshlak. Structure and complexity of planning with unary operators. *JAIR*, 18:315–349, 2003.

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

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

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

M. Helmert. Complexity results for standard benchmark domains in planning. *Art. Int.*, 143(2):219–262, 2003.

J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

J. Hoffmann. Local search topology in planning benchmarks: An empirical analysis. In *Proc. IJCAI-01*, pages 453–458, 2001.

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

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

K. Marriot and P. Stuckey. *Programming with Constraints*. MIT Press, 1999.

D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. AAAI-91*, 1991.

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

J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proc. ECP-01*, 2001.

J. Rintanen. A planning algorithm not based on directional search. In *Proc. KR-98*, pages 617–624, 1998.

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

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

V. Vidal and H. Geffner. Branching and pruning: An optimal temporal POCL planner based on constraint programming. In *Proc. AAAI-04*, pages 570–577, 2004.

V. Vidal and H. Geffner. Branching and pruning: An optimal temporal POCL planner based on constraint programming (long version). Technical report, 2004.

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

L. Zhu and R. Givan. Heuristic planning via roadmap deduction. In *4th. Int. Planning Competition Booklet (ICAPS-04)*, 2004.