Inference and Decomposition in Planning Using Causal Consistent Chains

Nir Lipovetzky
Departament of Technology (DTIC)
Universitat Pompeu Fabra
08018 Barcelona, SPAIN
nir.lipovetzky@upf.edu

Hector Geffner
Departament of Technology (DTIC)
ICREA & Universitat Pompeu Fabra
08018 Barcelona, SPAIN
hector.geffner@upf.edu

Abstract

Current state-of-the-art planners solve problems, easy and hard alike, by search, expanding hundreds or thousands of nodes. Yet, given the ability of people to solve easy problems and to explain their solutions, it seems that an essential inferential component may be missing. The reasons expressed by people for selecting actions appear to be related to causal chains: sequences of causal links $a_i \rightarrow p_{i+1}$, $i = 0, \ldots, n-1$, such that a_0 is applicable in the current state, p_i is a precondition of action a_i, and p_n is a goal. Some of these causal chains or paths appear to be good, some bad, others appear to be impossible. In this work, we focus on such paths and develop three techniques for performing inference over them from which a path-based planner is obtained. We define the conditions under which a path is consistent, provide an heuristic estimate of the cost of achieving the goal along a consistent path, and introduce a planning algorithm that uses paths as decomposition backbones. The resulting planner, called C3, is not complete and does not perform as well as recent planners that carry extensive but extremely efficient searches such as LAMA, but is competitive with FF and in particular, with FF running in EHC mode which yields very focused but incomplete searches, and thus provides, a more apt comparison. Moreover, many domains are solved backtrack-free, with no search at all, suggesting that planning with paths may be a meaningful idea both cognitively and computationally.

Introduction

Consider a simple instance of Blocks-World where n blocks, $1, \ldots, n$ on the table must be arranged into a single tower with block i on top of block $i+1$, $i = 1, \ldots, n-1$. In the initial state, there is a single best action, namely, picking up block $n-1$. Yet there are $n-1$ actions applicable in this state, and they all result in states with the same heuristic value according to either of the h_{add}, h_{max}, h_{FF}, or h^+ heuristics (Bonet and Geffner 2001; Hoffmann and Nebel 2001). In problems such as this, two approaches have been pursued in the context of heuristic search planners that are not exclusive of each other. One is the formulation of heuristics that take deletes into account; the other, is the use of implementations and search algorithms that allow more states to be evaluated before committing to the first action in a plan. The Fast Downward planner, for example, adopts both strategies, managing to integrate the notion of 'helpful actions' developed in FF in a complete search strategy (Helmert 2006). More recently, LAMA has built on this strategy, while switching back to delete-relaxation heuristics (Richter and Westphal 2008).

There is no question that more extensive searches and better heuristics are necessary for solving hard combinatorial problems, yet problems such as the one above, called Tower-n in (Vidal and Geffner 2005), are not hard, and the same can be said about many of the benchmarks in planning. This does not mean that 'easy' problems are easy for a domain-independent planner; the challenge is to recognize and exploit the structure that makes those problems easy by domain-independent methods, something that does not appear to be simple at all. Yet people appear to do that, while being able to explain their solutions, even if they are not particularly good at solving hard problems. In a problem such as Tower-n, they can immediately see that picking up a block other than $n-1$ is a wasted move. The reason for this, indeed, is not *heuristic* but *structural:* picking up a block m different than n-1 appears relevant to the goal through the 'path'

$$pick(m) \rightarrow hold(m) \rightarrow stack(m, m+1) \rightarrow on(m, m+1)$$

yet it can be formally proved that if this path is understood as a *sequence of causal links* (Tate 1977), no plan for achieving the goal can comply with it.

In this paper, we take the view that paths made up of sequences of causal links that reach a goal provide the reasons for selecting actions. However, some of these paths appear to be good, some bad, and still others, as illlustrated in the example above, are impossible. In this work, we focus on the study of such paths and develop three techniques for performing inference over them that result in a path-based planner called C3 for 'causal consistent chains'. We define the conditions under which a path is *consistent*, provide an *heuristic estimate* of the cost of achieving the goal *along a consistent path*, and introduce a *planning algorithm* that uses *paths as decomposition backbones.* The resulting planner is not complete and does not perform as well as recent planners that carry extensive but efficient searches such as LAMA, but is competitive with FF and in particular, with FF running in EHC mode which yields very focused but incomplete searches, and thus provides, a more apt comparison.

Moreover, many domains are solved backtrack-free, with no search at all, suggesting that planning with paths may be a meaningful idea both cognitively and computationally.

The paper is organized as follows. We provide first a background (Section 2) and then consider the consistency and minimality of paths (Sections 3 and 4), a planning algorithm that uses paths as decomposition backbones (Section 5), and a heuristic used to rank paths in the decomposition (Section 6). We then analyze the behavior of the C3 planner on two concrete examples (Section 7), and compare C3 experimentally with FF and LAMA over many IPC domains (Section 8).

Background

We consider Strips planning problems P = $\langle F, I, O, G\rangle$ where F is a set of fluents or atoms, $I \subseteq F$ and $G \subseteq F$ are the initial and goal situations, and O is a set of (ground) actions or operators a, each with an Add, Delete, and Precondition list $Add(a)$, $Del(a)$, $Pre(a)$. For convenience, and without loss of generality, we assume as in partial order planning that O contains an End action whose preconditions are the real goals of the problem and whose only effect is a dummy goal g so that $G = \{g\}$.

Most forward-state planners use heuristic functions $h(s)$ for guiding the search in the graph. The planner HSP uses the additive heuristic $h(s) = h(g)$, where $h(p) = 0$ for atoms p, if $p \in s$, and else is $h(p) = \min_{a \in O(p)}[1 + h(a)]$, where $O(p)$ is the set of actions in O that add p and $h(a) = \sum_{q \in Pre(a)} h(q)$ (Bonet and Geffner 2001). The max heuristic h_{max} is defined in a similar way but with the addition replaced by maximization. The max heuristic is equivalent to the heuristic that can be obtained from a relaxed planning graph (Hoffmann and Nebel 2001) by assigning to each fluent p or action a the index of the lowest layer where it appears, starting from layer 0.

The *best supporters* of a fluent $p \notin s$ in either heuristic, are the actions $a \in O(p)$ with smallest h. The heuristic $h_{\text{FF}}(s)$ used in FF is given by the size $|\pi_{\text{FF}}(s)|$ of the relaxed plan computed by FF in s. This plan $\pi_{\text{FF}}(s)$ can be defined recursively in terms of the best h_{max} supporters, by collecting backwards from the goal, a best supporter a for each goal, and recursively, a best supporter for each precondition of a that is not in s (Keyder and Geffner 2008). The definition in (Hoffmann and Nebel 2001) uses instead a relaxed planning graph and NO-OPs, along with a preference for NO-OPs supporters, which amounts to a preference for best (h_{max}) supports.

In addition to the heuristic, FF introduced two ideas that account to a large extent for its remarkable speed: *helpful action pruning (HA)* and *enforced hill-climbing search (EHC)*. Helpful action pruning is a criterion for eliminating from consideration a number of actions a in a state s without having to evaluate the heuristic for the resulting states s_a. The EHC search looks for a state s' that improves the heuristic of the current state s by carrying a breadth-first search from s, while pruning actions that are not helpful.

The notion of paths considered in this paper builds on the notion of *causal links* developed in the context of partial order planning (Tate 1977; McAllester and Rosenblitt 1991). A causal link a, p, b is a triple that states that action a provides the support for precondition p of b. This is taken as a *constraint* that implies that a must precede b in the plan and no other action that adds or deletes p can appear between them. We will see below that by exploiting this semantics of causal links, we will be able to propagate information along sequences of causal links $a_0, p_1, a_1, p_2, a_2, \ldots, p_n, a_n$ and show that some of such sequences are impossible. For this, we will make use of the notion of *(structural) mutexes*: pairs of atoms that cannot be both true in any reachable state and which can be computed in polynomial time (Blum and Furst 1995). More precisely, pairs $\langle p, q\rangle$ are mutex if $h^2(\langle p, q\rangle) = \infty$ where h^2 is a heuristic closely related to the heuristic underlying Graphplan, introduced in (Haslum and Geffner 2000). Provided with the mutexes, it is simple to show that a causal link a, p, b must rule out from the interval between a and b not only the actions that delete p but also the actions that do not add p and have a precondition q that is mutex with p. We say that actions *e-delete* p, either because they delete p, or because they presume that p is false and do not make it true (Nguyen and Kambhampati 2001; Vidal and Geffner 2005).

Paths: Consistency

We assume that the reasons for performing an action take the form of a sequence of causal links, that we call *causal chains*. When these causal chains reach the goal, we call them paths.

Definition 1 *A sequence a_0, p_1, a_1, p_2, a_2, ..., p_n, a_n of actions a_i and fluents p_i forms a* causal chain *if p_{i+1} is a precondition of action a_{i+1} and a positive effect (add) of action a_i for $i = 0, \ldots, n$.*

Definition 2 *A path is a causal chain that ends with the End action.*

Paths are not plans but rather *plan skeletons*, where actions may have to be filled in for achieving *all* the preconditions of the actions in the path.

A path starting with an action a that is applicable in the state s is said to be applicable in s. Such a path can be taken to suggest that a may be relevant for achieving the goal in s (Nebel, Dimopoulos, and Koehler 1997). Still, as argued above, it can be shown that certain paths, understood as sequences of causal links, cannot occur in any plan. We have seen an example in Tower-n, where blocks 1, ..., n intially on the table are to be arranged so that i is on top of $i + 1$ for $i = 1, \ldots, n - 1$. For this problem, the paths

$$t : pick(k), hold(k), stack(k, k + 1), on(k, k + 1), End$$

for any $k \neq n - 1$ can be shown not to be true in any plan.[1] Indeed, if there is any plan where the path holds, one can

[1] A causal link a, p, b is true in a sequential plan when a precedes b in the plan, a adds p, p is a precondition of b, and no action between a and b either adds or deletes p. If there are many occurrences of a and b in the plan, then the CL is true in the plan when is true for one pair of such occurrences. It is direct to extend this to sequences of causal links being true in a plan.

show that $ontable(k+1)$ will be true when End is executed, but this atom is mutex with the precondition $on(k+1, k+2)$ of End because $ontable(k+1)$ is initially true and remains true when $hold(k)$ is preserved (first causal link), and when $on(k, k+1)$ is preserved (second causal link). This sort of inference, that captures *side-effects* along chains is formalized below.

We can characterize the sets of fluents that must be true *before* and *after* applying each action a_i in a causal chain t, for *any plan* complying with the chain, in terms of the sets $F_t^-(a_i)$ and $F_t^+(a_i)$, defined as follows:

Definition 3 *The sets $F_t^-(a_i)$ and $F_t^+(a_i)$, for a causal chain $t : a_0, p_1, a_1, \ldots, p_n, a_n$ applicable in a state s, $0 \leq i \leq n$, are*

- $F_t^-(a_0) = s$
- $F_t^+(a_i) = Update(a_i; F_t^-(a_i))$
- $F_t^-(a_{i+1}) = Persist(p_{i+1}; F_t^+(a_i))$

where $Update(a; s)$ is the set of facts

$$(s \cup Pre(a) \cup Add(a)) \setminus Del(a)$$

and $Persist(p; s)$ is the maximal subset $Q \subseteq s$, *such that for every $q \in Q$, all the actions a that delete q, either add or e-delete p, or have a precondition that is mutex with a $q' \in Q$.*

This definition does not provide a complete characterization but can be computed efficiently in low polynomial time. The only subtlety is in the computation of $Persists(p; s)$ that must be done iteratively, in no more iterations than fluents in the problem, and usually much fewer.

As an illustration, the forward labels F_t^- and F_t^+ that are obtained for the path t above, for any block $k \neq n-1$ are

- $F_t^-(a_0) = \{ontable(j), clear(j), armfree\}$
- $F_t^+(a_0) = \{ontable(i), clear(i), hold(k)\}$
- $F_t^-(a_1) = \{ontable(i), hold(k)\}$
- $F_t^+(a_1) = \{ontable(i), on(k, k+1), clear(k), armfree\}$
- $F_t^-(a_2) = \{on(k, k+1), ontable(k+1)\}$

for each of the 3 actions a_i in t (pick, stack, End), and i and j ranging over $[1..n]$ with $i \neq k$. Thus, as mentioned above, $ontable(k+1)$ gets propagated until the action End, and since this atom is mutex with the precondition $on(k+1, k+2)$ of End (one of the goals), it suffices to prove that the path is not feasible in any plan.

The definition above captures side effects by reasoning forward along the direction of a causal chain. It is also possible to infer necessary side effects by reasoning backwards. The definition of the backward labels $B_t^+(a_i)$ and $B_t^-(a_i)$ proceeds in an analogous way:

Definition 4 *The sets $B_t^-(a_i)$ and $B_t^+(a_i)$, for a causal chain $t : a_o, p_1, a_1, \ldots, p_n, a_n$ $0 \leq i < n$, are*

- $B_t^-(a_n) = Pre(a)$
- $B_t^+(a_i) = PersistB(p_{i+1}; B_t^-(a_i + 1))$
- $B_t^-(a_i) = UpdateB(a_i; B_t^+(a_i))$

where $UpdateB(a; s)$ is the set of facts

$$(s \cup Pre(a)) \setminus Add(a))$$

and $PersistB(p; s)$ is the maximal subset $Q \subseteq s$, *such that for every $q \in Q$, all the actions a that add q, either add or e-delete p.*

Indeed, in the same way that the atom $ontable(k+1)$, for $k \neq n$, can be propagated forward along the path t into the set $F_t^-(End)$, the atom $on(k+1, k+2)$ can be propagated backwards from $Pre(End)$ into $B_t^-(pick(k))$. This is because the only action that adds $on(k+1, k+2)$, namely $stack(k+1, k+2)$, e-deletes $on(k, k+1)$, as its precondition $clear(k+1)$ is mutex with this atom, while it also e-deletes $hold(k)$. The atom $on(k+1, k+2)$ in $B_t^-(pick(k))$ thus means that in any plan complying with the chain t, the atom $on(k+1, k+2)$ must be true just before the first action of the chain; an inference that can be understood as a form of goal ordering along a chain (Koehler and Hoffmann 2000). We will indeed refer to the fluents p in a label $B_t^-(a_i)$, such that $p \notin Pre(a_i)$, as the *implicit preconditions* of the action a_i along the chain t. These conditions are needed at the time a is executed, not by a, but by the actions that follow a along the path.

Forward and backward inference along a chain can be combined to interact synergistically. For example, knowing that a certain fluent p must persist backwards along an interval, can help to establish that a certain other fluent q must persist forward along the same interval, if the actions that delete q, e-delete p. This combination of forward and backward inference yields fixed point labels that we will refer to as the final labels $L_t^-(a_i)$ and $L_t^+(a_i)$. The consistency of a path is then defined as follows:

Definition 5 *A causal chain applicable in a state s is* inconsistent *in s if one of the final labels along the chain include a mutex pair. If the chain is not inconsistent, it is said to be consistent.*

This is a sound but incomplete definition that can be verified in polynomial time:

Proposition 6 *An inconsistent causal chain applicable in a state s cannot be true in any plan from s.*

The notion of *implicit preconditions* along a path and the notion of *path consistency*, will be two of the building blocks of the planning algorithm below.

Minimality

A path t starting with the action a in a state s indicates that the action a appears to be relevant to the goal. If t is inconsistent, this relevance is only apparent; the path carries no weight. There are however, paths that are consistent and yet, should be discarded as well. Consider again the initial state of the Tower-n and the path

$$t' : pick(i), hold(i), stack(i, j), on(i, j), unstack(i, j), \\ hold(i), on(i, i+1), End$$

where, after block i picked up, it is placed on top of another block $j \neq i+1$. This path contains the irrelevant action $unstack(i, j)$: irrelevant because it is used to get the atom $hold(i)$ which appears earlier in the path supported by the action $pick(i)$.

Algorithm 1 $SOLVE(s, t, K, \pi)$: *C3* Planning Algorithm

Input: initial state $s = s_0$; initial path $t = \{End\}$; atoms to keep $K = \{\}$; plan prefix $\pi = \{\}$

```
1: G ← Prec of first action in t
2: if G true in s then
3:   if t = End then                                   ;; Plan found
4:     Return(π)
5:   else                                              ;; Reduce
6:     SOLVE(s', t', K', π + a)
7:       where t = a, p, t'
8:         s' = do(a, s)
9:         K' = (K − ∀(q, a) ∈ K) + (p, b)
10:        s.t. b is the first action in t'
11:  end if
12: else                                               ;; Extend
13:   COMPUTE Min Graph from s excluding actions that e-
      delete q for (q, b) ∈ K
14:   CHOOSE Min Path a, p, t1 for G in min graph that is con-
      sistent, computing implicit preconditions
15:   SOLVE(do(a, s), t1 + t, K + (p, b), π + a)
16:   if no consistent path left then
17:     if if t = {End} and K = {} then
18:       fail                                         ;; Backtrack
19:     end if
20:   else                                             ;; Reset
21:     SOLVE(s, {END}, K = {}, π)
22:   end if
23: end if
```

A simple way to prune such spurious paths from consideration is by requiring the actions a_i that support the fluents p_{i+1} to be among the *best supporters* for p_{i+1}, which as defined above, are the actions a that add p_{i+1} and have min $h_{max}(a)$ value. We call the resulting paths, minimal paths:

Definition 7 *A minimal path in the state s is a path a_0,p_1, a_1,p_2,a_2,. . .,p_n,a_n applicable in s where each action a_i, for $i = 0, \ldots, n-1$, is a* best supporter *of p_{i+1} in s.*

Minimal paths are implicit in the definition of helpful action pruning in FF. If we say that an action a is *minimal* in state s when there is a *minimal path* starting with $a_0 = a$ in s, it follows that all helpful actions in s are minimal. The converse relation, however, does not hold. The minimal actions can be shown to be actually the actions that are helpful in *some* relaxed plan; where different relaxed plans are obtained according to the way ties among best h_{max} supporters are broken.

Minimal paths in a state s can be computed efficiently from the *minimum graph* obtained from the h_{max} heuristic in s by tracing backward from the goal all their best supporters and the best supporters of their preconditions recursively. While the minimal paths in the min graph can be exponential in number, this turns out to be seldom the case.

In the planner C3, the only paths considered are minimal paths. This is what makes the planner incomplete, in the same way that the EHC search over the helpful actions is incomplete in FF. There is a difference though: while FF computes the helpful and hence minimal actions without restriction, C3 computes the minimal actions in the context of a set of commitments. Then the minimal actions are not necessarily the actions that are minimal when no commitments are made.

We will say that a path t is minimal in a state s in the context of a set of atoms p that must be preserved, if t is minimal in the modified problem where the actions that delete or e-delete p are excluded. Such paths are obtained from the min graph obtained from the h_{max} heuristic with the exclusion of those actions.

Decomposition: The Planning Algorithm

The plan algorithm in C3[2] regards a consistent path $t : a_0, p_1, a_1, \ldots, p_n, a_n$ in a state s as a recipe for a *decomposition* where the actions a_0, a_1, ...are executed in that order by solving a series of subproblems: after applying the action a_i, the preconditions of the next action a_{i+1} become the goal, and so on, until reaching the End action a_n. In this decomposition, the *implicit preconditions* of each action a_i in the path, derived backward from the goal, are included among the explicit preconditions $Pre(a)$. A planning problem is solved by C3 by finding first a consistent path to decompose the problem, and using the same method recursively to decompose the first subproblem until the path can be reduced.

We refer to the operation of applying the action a_i after its (explicit and implicit) preconditions have been achieved, as a *reduction* step. In the reduction of a path $a_i, p_{i+1}, a_{i+1}, \ldots, p_n, a_n$, the current state is progressed through the action a_i, and the algorithm is then called on the subproblem with path $a_{i+1}, \ldots, p_n, a_n$, where the atom p_i is *maintained* until the action a_{i+1} is executed.

A reduction is not possible when the preconditions of the first action do not hold in the current state s. If $t : a_i, p_{i+1}, a_{i+1}, \ldots, p_n, a_n$ is the path then, a new path t' is created by composing a minimal chain for an 'open' precondition of a_i (that is not true in s), and the 'tail' t. This operation is called an *extension*. For the construction of the minimal chain, the actions that delete or e-delete conditions that must be maintained until a_i is executed, are excluded.

The paths that are created by the *extensions* are always checked for consistency: inconsistent paths are skipped over while the implicit preconditions are computed for the paths found to be consistent.

The state of the planning algorithm or solver is a tuple comprising the current state s, the committed path t , the set K of pairs $\langle p, a\rangle$ so that p needs to be preserved until a is applied, and the plan prefix π. Initially, $s = s_0$, $t = \{End\}$, and $K = \pi = \{\}$.

The reduction and extension steps modify the state of the solver that exits successfully when the tail $t = \{End\}$ is reduced. On the other hand, when the current path t cannot be reduced or extended with a consistent path, the commitments, i.e., t and K, are *reset*. The option here is to backtrack rather than to reset, yet this option produces weaker empirical results. The solver backtracks though when there is nothing to reset; namely, when the solver state is a reset

[2]This is C3v2.0; a prior version of the planner using a slightly different algorithm, entered the 2008 IPC, where it obtained the 'Jury Prize'.

state in which the path cannot be reduced or extended consistently.

Pseudo-code for the C3 planning algorithm is shown in Figure 1. The minimal chains in the extend operation are computed very much like relaxed plans, in two phases: in the forward phase, the heuristic $h_{max}(p)$ is computed for all fluents p from the current state s; in the second phase, all the best h_{max} supporters of the goals are marked, and the process is applied recursively to the preconditions of such actions, excluding the goals and preconditions that are true in s. In this process, actions a that e-delete an atom p that must be preserved at that stage, i.e. atoms p for a pair (p, b) in K, are excluded. The minimal chains compatible with the commitments in K that extend the current path t can then be obtained starting with the actions applicable in s that are marked. The CHOOSE construct in the algorithm expresses a non-deterministic choice that is critical for the actual performance of the algorithm: the order in which consistent minimal paths are constructed and considered, given by the best supporter marks. We address this issue next.

There are two sources of incompleteness in the algorithm. First, the exclusion of paths that are not minimal in the extension operation, and second, the interpretation of paths as sequential decompositions. Neither choice, however, turns out to be as restrictive as it may appear. In the first case, because the minimal paths are computed given a set of commitments; in the second, because the implicit preconditions take into account not only the immediate goals, given by the preconditions of the first action in the path, but further goals down the path as well.

Preferences

The building block for ordering the paths in the extension step, is a new heuristic $h(t|s)$ built from a known base heuristic, that estimates the cost of achieving the goal *along the path* t. The interesting thing about this new heuristic is that it takes deletes into account, even if the base heuristic doesn't, and it penalizes paths t that provide bad decompositions. Moreover, if the sequence of actions in t constitutes a plan from s, only then, $h(t|s) = 0$, as the heuristic excludes the cost of the actions already committed (actions in t). The base heuristic is the additive heuristic, but other heuristics could be used too.

Let t be the consistent path $a_1, p_2, \ldots, p_n, a_n$. This path does not have to be applicable in the state s, but as any path, it must reach the goal (i.e. $a_n = End$). For any such path, we provide first an (heuristic) estimation of the state s_{i+1} that results right after the action a_i in the path is applied. We use the expression $\pi(a_i; s_i)$ to denote a relaxed plan that achieves the (explicit and implicit) preconditions of action a_i in the state s_i along the path. This relaxed plan is obtained by collecting the best supporters according to the base heuristic, backwards from the goal (Keyder and Geffner 2008).

If $\pi_i = \pi(a_i, s_i)$ is the relaxed plan for achieving the preconditions of a_i from s_i, then the state s_{i+1} *projected* after applying the action a_i is estimated as

$$s_{i+1} = (((s_i \setminus Del(\pi_i)) \cup Add(\pi_i)) \setminus eDel(a_i)) \cup Add(a_i)$$

where $Add(\pi_i)$ is the set of fluents added by the actions in π_i, $eDel(a_i)$ is the set of fluents e-deleted by the action a_i and $Del(\pi_i)$ refers to a *subset* of the fluents deleted by actions in π_i. This subset is defined as the fluents that are deleted not just by one action in π_i, that is a best supporter of some fluent p in the the relaxed plan, but by *all* the best supporters of p, whether they made it in the relaxed plan or not. The reason is that the choice of best supporters in the relaxed plan is rather arbitrary, and deleting a fluent because an arbitrary best supporter deletes it turns out to be more critical than adding a fluent that an arbitrary supporter adds. So these deletions aim to be cautious.

A state sequence $s_1, \ldots, s_n$ is then generated for a consistent chain $t : a_1, p_2, \ldots, p_n, a_n$ in a state s, according to the formula above by setting s_1 to s, $\pi(a_i, s_i)$ to the relaxed plan for obtaining the preconditions of a_i from s_i, and s_{i+1} as in the formula above. This sequence is used to compute the heuristic $h(t|s)$, that estimates the cost of achieving the goal along the path t and can be expressed as

$$\sum_{i=1}^{n} h(Pre(a_i)|s_i)\,,$$

where recall that implicit preconditions in the path are treated as action preconditions. This estimate is just the sum of the estimated costs of solving each of the subproblems along the path, assuming that the states s_i along the path are those in the projected state sequence.

A problem with this estimate, however, is that due to the use of deletes, it's often infinite. This may reflect that the projected state sequence is misleading, but more often, that the decomposition expressed by the path t is not perfect. For example, if a precondition p of an action a_i cannot be established from the state s_i, yielding $h(a_i|s_i) = \infty$, it is possible that such a precondition can be established in the previous subproblem from the state s_{i-1} and maintained into the following subproblem if the action a_{i-1} does not e-delete it.

With this in mind, we define the estimated cost of achieving the goal through a consistent path $t : a_1, \ldots, p_n, a_n$ as

$$h(t|s) = \sum_{i=1}^{n} h_i(Pre(a_i)|s_i)\,,$$

where h_1 is equal to the base heuristic h, and h_{i+1} is

$$h_{i+1}(p|s_{i+1}) = \min\left[\, h(p|s_{i+1})\,,\; h_i(p|s_i) + \Delta_i(p)\,\right]$$

where $\Delta_i(p)$ is a penalty term for bringing p from the subproblem i along the path t to subproblem $i+1$. We have set $\Delta_i(p)$ to a large constant, independent of i and p (10 in our experiments), except when the action a_i e-deletes p where $\Delta_i(p)$ is set to ∞. In the computation of the base heuristic $h(p|s_i)$ for all fluents p in the subproblem that corresponds to $s = s_i$, all the actions that e-delete a fluent in the label $L_t^-(a_i)$ are excluded, as those are the fluents that must hold prior to a_i in any plan that complains with the path t.

Provided this heuristic function, the extensions $t' = b_1, q_1, b_2, \ldots, q_m, b_m$ of a path t in a state s in the planning algorithm, are constructed incrementally, starting with the actions that are applicable in s that have been marked as best

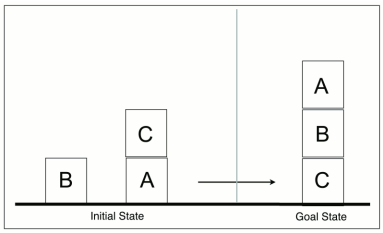

Figure 1: A blocks-world instance requiring goal interleaving

supporters in the extension step in the algorithm. The action b_1 is chosen as the action that minimizes $h(t|s_1)$ where s_1 is the state that results from applying b_1 in s, and given the action b_i and the state s_i projected along the t' path, the action b_{i+1} is chosen as the one that minimizes $h(t|s_{i+1})$, among the actions in the min graph with a precondition q_i that is added by b_i.

In these extensions, we prune the chains that contain actions b_i that either appear already in one of the relaxed plans $\pi(b_k, s_k)$, for some $k < i$, or that support an atom q_i in the path that some action in those relaxed plans add. This is because when the planner commits to a path like $b_1, q_1, b_2, q_2, \ldots$ where b_i is a best supporter for q_i, it is reasonable to assume that in the plans that comply with this path, b_i and q_i do not occur prior to their occurrence in the path. In other words, the assumption of best supporters in the path, is also an assumption of first supporters in the plans complying with the path.

Examples

The C3 planner is supposed to be a transparent planner where the choice for the actions selected can be explained in terms of reasons given by paths. Thus, before reporting the tables that are standard in the experimental evaluation of planners, we analyze the behaviour of C3 over two examples.

Blocks World

The so-called Sussman Anomaly is a small instance of the Blocks-World domain known because it requires some form of goal interleaving. The instance is shown in Figure 1: the problem has two goals, b on c, and a on b, but no goal can be tackled first while leaving the other goal aside; the subgoals of the two problems need to be interleaved, something that appears to defy the decomposition in the C3 planner.

The first consistent path that extends the initial path given by the End action, in the initial state, is

$$t_1 : unstack(c, a), clear(a), pick(a), hold(a), \\ stack(a, b), on(a, b), End$$

with the goal $on(b, c)$, which is the other precondition of End, inferred to be an implicit precondition of $pick(a)$, as the actions that add $on(b, c)$ e-deletes both $hold(a)$ and $on(a, b)$. This is the second path considered in the extension step, because the first path

$$t_2 : pick(b), hold(b), stack(b, c), on(b, c), End$$

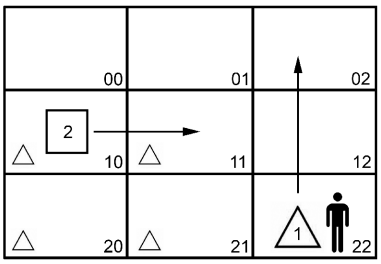

Figure 2: A Grid instance: key 1 needed to unlock positions 11 and 10 to reach key 2. Key 2 must be placed at 11 and key 1 at 02.

is found to be inconsistent: the precondition $on(a, b)$ is mutex with the atom $ontable(a)$ that is propagated along the path from the initial state to the End action.

Once the path t_1 is returned, it is then reduced by applying its first action $unstack(c, a)$ in the initial state $s = s_0$. This reduction implies a commitment to the fluent $clear(a)$ until $pick(a)$ is executed, and a new path t'_1 given by the tail of t_1 starting with $pick(a)$. This new path cannot be reduced because $pick(a)$ has explicit and implicit preconditions that do not hold in the resulting state s_1 where block c is being held. The planner then extends t'_1 with the chain

$$t_3 : putdown(c), handfree, pick(b), hold(b), \\ stack(b, c), on(b, c)$$

that supports one of open (implicit) preconditions of $pick(a)$ and yields the consistent path $t_3 + t'_1$. It turns out that the actions in this path form a plan from the state s_1, and thus, the problem is solved after 6 successive reductions. Since no more extensions are needed, the problem is solved backtrack-free in two extensions: the first that generated the path t_1 that extended the initial path $t = \{End\}$; the second, that generated the path $t_3 + t'_1$ that extended the tail t'_1 of t_1.

Grid Problem

The Grid domain consists of an agent, that can carry a key, moving around in a grid. Some of the cells in the grid are locked and can be opened by keys of a certain shape. Before moving into a locked place, however, the agent has to unlock it from an adjacent position. The lower-right numbers are the cell indexes in the grid, and the lower-left shapes indicate if the positions are locked and the shape needed to unlock them. The square at position 10 is the shape of key 1 ($k1$) and the triangle is the shape of key 2 ($k2$) at position 22. The arrows indicate where the keys have to be moved (Fig. 2).

Before finding the first consistent path from the initial state s_0 in the instance shown , the following paths are generated

$$t_1 : pick(22, k1), hold(k1), put(02, k1), k1@02, End.$$
$$t_2 : pick(22, k1), hold(k1), put(11, k1), k1@11, \\ pick\&lose(11, k1, k2), k2@11, End.$$

and pruned, although this time not because they are inconsistent but because both have heuristic values $h(t_i|s_0) = \infty$. In the first case, because of the last subproblem, where

the precondition $k2@11$ must be achieved from a projected state where $k2$ is at the (still) locked position 10, while the atom $k1@02$ is preserved. In the second, because of the subproblem where the precondition $hold(k2)$ of $pick\&lose(11, k1, k2)$ must be achieved from a projected state where $k2$ is at the locked position 10 while maintaining the atom $k1@11$. Actually, the generation of the path t_2 is pruned at that point and never reaches the End action.

The third path generated turns out to be consistent and is

$$t_3 : pick(22, k1), hold(k1), unlock(12, 11, k1), \\ open(11), move(12, 11), atrobot(11), \\ unlock(11, 10, k1), open(10), move(11, 10), \\ atrobot(10), pick\&lose(10, k2, k1), hold(k2), \\ put(11, k2), k2@11, End,$$

that describes a *plan skeleton* where after picking up $k1$, the robot unlocks 11 from 12, moves then to 11, unlocks 10, moves then to 10, exchanges $k1$ with $k2$, and places $k2$ at its target cell. This is a plan skeleton as some actions need to be filled in by suitable extensions. Indeed, this path can be reduced up to End after two successive and successful extensions calls, involving the action $move(22, 12)$ before $unlock(12, 11, k1)$, and the action $move(10, 11)$ before $put(11, k2)$.

After this reduction, the tail contains the End action only, that cannot be reduced since its other precondition $k1@02$ does not hold, and then an extension call that keeps the atom $k2@11$, results in the following consistent path:

$$t_4 : move(11, 10), atrobot(10), pick(10, k1), \\ hold(k1), put(02, k1), k1@02, End$$

whose first two actions can be reduced right away, and where the 'open' precondition $atrobot(02)$ of the third action, $put(02, k1)$ results in an extension call that fills in the moves needed to apply the action and get to the goal. The plan is thus obtained after 3 main extensions, the first one extends the initial path that just contains the End action and results in the path t_2, the second that extends the initial path but in a different state and with the commitment to keep $k2@11$, and the third that extends the tail $put(02, k1), k1@02, End$ of t_4 with the move actions to get to $atrobot(02)$.

Experimental Results

We compare C3 with FF and LAMA over a broad range of planning benchmarks. C3 is written in C++ and uses Metric-FF as an *ADL* to *Propositional STRIPS* compiler. (Hoffmann 2003). LAMA is executed without the plan improvement option, reporting the first plan that is found. All experiments were conducted on a dual-processor Xeon 'Woodcrest' running at 2.33 GHz and 8 Gb of RAM. Processes time or memory out after 2 hours or 2 Gb. All action costs are assumed to be 1 so that plan cost is plan length.

Table 1 compares C3 with *FF* and *LAMA* over 545 instances from previous IPCs. In terms of coverage, C3 solves 5 less problems than LAMA but 14 more than FF, and more remarkably, it solves 452 problems (30 less than FF and 28 more than FF in EHC) *backtrack-free*, that is, heading directly to the goal, without ever having to revise a plan prefix. There are several domains where C3 solves more problems than both FF and LAMA, the largest difference being in Storage, where C3 solves 29 problems, and FF and LAMA 18. On the other extreme, FF and LAMA solve all the 20 FreeCell instances, while C3 solves only 6.

A measure of the number of choices made in FF and LAMA is given by the number of nodes expanded. In C3, this measure is given by the number of extension operations. As we have seen, the number of extension operations can be smaller than the number of actions in the plan, as often, many of the actions in a path can be reduced one after the other. While in many cases, FF and LAMA expand thousands of nodes, C3 appear to solve a similar number of problems by performing a much reduced number of extensions. This, however, does not mean that C3 is faster; actually, it is not, due to the overhead in the ordering, filtering, and selection of paths.

The average quality of the plans found by C3 is 18% worse than those found by FF, and 9% worse than those found by *LAMA*, with the largest differences in *Miconic* and *Pipesworld*. In some cases, however, it delivers shorter plans, like in *Depots*, where it delivers plans that are 10% shorter.

Table 2 compares C3 with *FF* and *LAMA* over 240 instances of the more recent IPC held in 2008. In this set of benchmarks, C3 does relatively worse, with LAMA solving 227 of the problems, FF solving 201, and C3 solving 188; 85% backtrack-free.

Domains like *M-Prime*, *Pathways*, *Pipes-NT*, and *Psr-Small*, are not included in the tables because one or more of the planners had problems parsing the instances. On the other hand, in *Sokoban*, C3 could not solve *any instance*, very much as FF in EHC mode. In both, the problem arises from the focus on the paths that are minimal. The version of C3 used in the last IPC solves these instances by triggering a best-first search when the basic algorithm fails, very much as FF. For clarity, however, we have excluded this option here.

Summary

We have focused on the study of paths for planning and developed three techniques for performing inference over them. The resulting planner is not complete and does not perform as well as recent planners, but surprisingly, does not lag far behind. Moreover, of the collection of 728 problems, 78% of them are solved backtrack-free. For the future, we want to study *learning* in the context of path-based planning, in the sense that is used in SAT: where the causes of failures are identified and used to prune the rest of the search. We also want to use paths to get a better understanding of hierarchical planning and the conditions under which it pays off. Finally, the path-based heuristic can be used for state-based planning and may subsume the notion of consistency provided that inconsistent paths result in infinite heuristic values.

		FF				$C3$					LAMA			
Domain	I	S	EHC	Expands	T	S	BF	Extends	T	Quality	S	Expands	T	Quality
Blocks World	50	42	42	9,193	0.22	50	50	23	8.79	114%	50	1,496	0.87	187%
Depots	22	22	19	57,777	44.08	22	22	23	107.82	90%	20	42,609	46.58	102%
Driver	20	16	6	7,062	41.28	20	17	21	41.02	118%	20	7,892	4.67	104%
Ferry	50	50	50	50	$<$ 1msec	50	50	17	0.07	117%	50	108	0.18	104%
Free Cell	20	20	14	5,573	38.80	6	3	28	26.21	139%	20	3,091	34.30	115%
Grid	5	5	5	301	$<$ 1msec	5	3	12	107.72	98%	5	174	5.58	95%
Gripper	50	50	50	102	$<$ 1msec	50	50	51	0.98	132%	50	80	0.00	100%
Logistics	28	28	28	94	$<$ 1msec	28	28	29	0.76	135%	28	97	0.25	101%
Miconic	50	50	50	52	$<$ 1msec	50	50	26	0.02	154%	50	37	0.15	100%
Mystery	30	19	15	75,031	323.08	23	14	4	2.44	103%	22	63,597	15.31	93%
Open Stacks	30	30	30	993	7.12	29	29	87	557.51	99%	30	162	12.80	102%
Pipes World	50	22	4	36,572	18.01	28	22	37	957.33	152%	28	36,140	112.25	118%
Rovers	40	40	40	10,341	26.97	39	39	42	548.88	102%	40	1,750	13.44	103%
Satellite	20	20	20	389	0.02	20	20	21	3.24	101%	20	412	0.90	107%
Storage	30	18	3	142,526	15.00	29	10	45	446.06	100%	18	3,348	1.62	121%
TPP	30	30	30	116,904	426.90	27	25	63	586.84	122%	30	1,805	13.11	87%
Zeno Travel	20	20	18	135	2.14	20	20	18	53.15	129%	20	482	3.55	117%
Total	545	482	424	27,241	78.63	496	452	32	202.87	Average	501	9,605	15.62	Average
Percentage		88%	78%			91%	83%			118%	92%			109%

Table 1: C3 vs. FF and LAMA on instances of previous IPCs: *S* is number of solved instances, *EHC* is number of problems solved by EHC, *Expands* is avg number of expanded nodes, *BF* is number of instances solved *Backtrack-free*, *Extends* is average number of path extensions, *T* is avg time in seconds, and *Quality* is plan quality ratio; e.g. 200% means plans twice as long as those reported by FF on avg.

		FF				$C3$					LAMA			
Domain	I	S	EHC	Expands	T	S	BF	Extends	T	Quality	S	Expands	T	Quality
Cyber	30	4	4	228	0.74	20	20	15	170.71	100%	25	151	402.26	100%
Elevator	30	30	30	1,254	1.34	30	29	61	226.25	125%	30	666	3.28	103%
Openstacks	30	30	30	866	0.59	30	30	78	180.84	100%	24	397	0.48	105%
Parc Printer	30	30	21	252	0.05	30	30	18	6.03	102%	30	13,980	3.53	104%
Pegsol	30	30	0	32	9.87	21	4	4,021	70.67	104%	28	873	25.84	100%
Scanalyzer	30	30	22	2,166	55.57	25	25	14	173.31	94%	30	4,882	41.02	97%
Transport	30	30	30	105,754	355.50	24	14	77	806.67	125%	30	6,828	9.83	85%
Wood	30	17	12	1,027	5.08	8	8	21	34.70	98%	30	146	3.68	104%
Total	240	201	149	13,947	53.59	188	160	538	208.65	Average	227	3,490	61.24	Average
Percentage	100%	84%	62%			78%	67%			106%	95%			100%

Table 2: C3 vs. FF and LAMA on instances from the 2008 IPC

Acknowledgements

This work is partially supported by grant TIN2006-15387-C03-03 from MEC/Spain.

References

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

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proc. of the Fifth International Conference on AI Planning Systems (AIPS-2000)*, 70–82.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

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

Hoffmann, J. 2003. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *J. Artif. Intell. Res. (JAIR)* 20:291–341.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In *18th European Conference on Artificial Intelligence (ECAI-08)*.

Koehler, J., and Hoffmann, J. 2000. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *JAIR* 12:338–386.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of AAAI-91*, 634–639.

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *Proc. ECP*, 338–350.

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

Richter, S., and Westphal, M. 2008. The LAMA planner: Using landmark counting in heuristic search. In *6th. Int. Planning Competition Booklet (ICAPS-06)*.

Tate, A. 1977. Generating project networks. In *Proc. IJCAI*, 888–893.

Vidal, V., and Geffner, H. 2005. Solving simple planning problems with more inference and no search. In *Proc. CP-05*.