Structural Inference in Planning: Scaling up without Heuristic Estimators

Nir Lipovetzky
Departamento de Tecnología
Universitat Pompeu Fabra
08003 Barcelona, SPAIN
nir.lipovetzky@upf.edu

Miquel Ramírez
Departamento de Tecnología
Universitat Pompeu Fabra
08003 Barcelona, SPAIN
miquel.ramirez@upf.edu

Hector Geffner
Departamento de Tecnología
ICREA & Universitat Pompeu Fabra
08003 Barcelona, SPAIN
hector.geffner@upf.edu

Abstract

The aim of this paper is to introduce a different approach to the problem of inference in planning that is not based on either the extraction and use of heuristic functions or reductions into SAT or CSPs. The proposed approach is based on the new notion of *consistent causal chains:* sequences of causal links a_i, p_{i+1}, a_{i+1} starting with an action a_0 applicable in the current state s and finishing in the goal, where p_{i+1} is an effect of action a_i and a precondition of action a_{i+1}. We show that by enforcing the semantics of causal links, it is possible to propagate side effects along such chains and detect that some of these chains cannot be part of any plan. Actions a_0 that cannot start any consistent causal chain can then be pruned. We show that while simple, this pruning rule is quite powerful: a plain backtracking forward-state search planner with a version of this pruning rule solves as many benchmark problems as the effective Enforced Hill Climbing search of FF, many of them backtrack-free. We then consider extensions of this basic idea and discuss some of its implications.

Introduction

The automatic derivation of informative heuristic functions has been a key development in modern domain-independent planning (McDermott 1999; Bonet and Geffner 2001). Heuristic functions provide the search for plans with a sense of direction that allows large problems to be solved quite effectively. The main virtue of heuristic search planners such as FF (Hoffmann and Nebel 2001) is that they are pretty robust: they manage to solve problems, even if the heuristic is not accurate, by evaluating thousands of nodes sufficiently fast. Their main limitation, on the other hand, is that they are not transparent: it is not clear why they work, it is not simple to explain, as humans do, why certain actions are selected and others are discarded, and most important of all, because of these reasons, it is not simple to improve them in spite of their known limitations. In fact, the FF planner developed 8 years ago and based on the delete-relaxation heuristic, still represents, to a large extent, the state of the art. [1]

[1]More recent planners, including the winner of the last two planning competitions, SGPlan (Chen, Wah, and C.Hsu 2006), does better than FF on some recent benchmarks but worse in others, and does not dominate FF according to our experiments.

The goal of this paper is to introduce a different approach to the problem of inference in planning that is not based either on the extraction and use of heuristic functions or reductions into SAT or CSPs. It is based rather on the new notion of *consistent causal chains:* sequences of causal links a_i, p_{i+1}, a_{i+1} starting with an action a_0 applicable in the current state s and finishing with the End action a_n, where p_{i+1} is an effect of action a_i and a precondition of action a_{i+1}. We show that by enforcing the semantics of causal links, it is possible to propagate side effects along such chains and detect that some of these chains cannot be part of any plan. Actions a_0 that cannot start any consistent causal chain can then be pruned. We show that while this pruning rule is simple, it is very powerful too: a plain backtracking forward-state search planner with a version of this pruning rule solves as many benchmark problems as the effective Enforced Hill Climbing (EHC) search of FF, many of them backtrack-free. This is quite remarkable as FF's EHC search owes its power to both helpful action pruning and an *heuristic estimator* obtained from the relaxed planning graph. Interestingly, a similar backtrack-free behavior is reported for the CPT planner (Vidal and Geffner 2006), resulting in this case from constraint propagation over a sophisticated CP formulation of POCL planning (McAllester and Rosenblitt 1991; Weld 1994).

The problem of determining whether there is a consistent causal chain $a_0, p_1, a_1, p_2, a_2, \ldots, p_n, a_n$ ending in the goal turns out to be intractable. For this reason and also because we are particularly interested in causal chains that do not contain irrelevant actions, we focus on the computation of the consistent causal chains where for each atom p_{i+1}, the action a_i is a *best supporter* of p_{i+1}, where the notion of best supporters is obtained from a simple reachability analysis. We show empirically that the computation of these 'minimal' consistent causal chains, that must be repeated in every state s in the backtrack search, can be carried out efficiently, and results in solution times that are not that different from FF's EHC.

The paper is organized as follows: we review first some background notions, and then introduce the notions of relevant, minimal, and consistent paths. We lay out then the basic formal properties of consistent paths, including their semantics and complexity, and the computation of the min-

imal paths that are consistent. Building on these notions and suitable extensions, we define two simple forward-state planners that rely on backtracking search and consistency-based pruning criteria. We then report empirical results for these planners and end with a brief summary.

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 $G = \{g\}$. Thus all plans for P must include the action End.

Forward-state planners search for plans by looking for a path in a directed graph that connects a single root node with a goal node. The nodes in the graph represent the $2^{|F|}$ possible states in P, with the root node being the state $s_0 = I$, and the goal nodes being the states that include the dummy goal g. Likewise, an edge labeled with an action $a \in O$ connects a node s with a node s' if a is applicable in the state s resulting in the state s'; i.e. if $Pre(a) \subseteq s$ and $s' = (s \cup Add(a)) \setminus Del(a)$.

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 $h(p) = 0$ if $p \in s$ and else $h(p) = \min_{a in O(p)}[1 + h(a)]$, 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 a maximization. The max heuristic is equivalent to the heuristic that is 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 .

The *best supporters* of a fluent $p \notin s$ in either heuristic, are the action $a \in O(p)$ 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 of the fluents p, starting with the dummy goal g: namely, $\pi_{\text{FF}}(s)$ contains the best supporter a_g of g, and for each action a in $\pi_{\text{FF}}(s)$, a best supporter for each precondition p of a not in s. 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.

The planner FF has been shown to scale up much better than HSP, with the heuristic used in FF, however, playing a relatively small role. In addition to the heuristic, FF introduced two ideas that accounts 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, in turns, looks iteratively 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. Helpful action pruning is not sound (it may render problems unsolvable) and the EHC search is not complete (it can miss the goal), yet together with the h_{FF} heuristic yield a powerful planner that constitutes the "basic architecture" of FF (Hoffmann and Nebel 2001). When this basic architecture fails, FF switches to a slower but complete search mode: a best-first search from the initial state where no actions are pruned.

The inference scheme for planning below is used in the context of a forward-state search but does not appeal to a heuristic function but to 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 that no other actions 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 the heuristic $h^2(\langle p, q\rangle)$, closely related to the heuristic underlying Graphplan, is infinite (Haslum and Geffner 2000), and they can be computed even more effectively by setting the costs of all actions to 0. 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 c that do not add p and have a precondition q that is mutex with p. We say in these cases that c *e-deletes* p (Nguyen and Kambhampati 2001; Vidal and Geffner 2006).

Relevant and Minimal Paths

We are interested in the conditions under which an applicable action in an arbitrary state s can be pruned. Helpful action pruning provides one such criterion but its justification is empirical; from a theoretical point of view sometimes it is too strong (rendering a problem unsolvable) and sometimes too weak (failing to prune actions that cannot help, as we will see). So we start with the more basic notion of *relevance* as captured by sequences of causal links a_i, p_{i+1}, a_{i+1} that we refer to as *causal chains*.

Definition 1 (Causal Chains) *A sequence a_0, p_1, a_1, p_2, a_2, …, p_n, a_n of actions a_i and fluents p_i is a* causal chain *in a state s if action a_0 is applicable in s, and fluent p_{i+1} is a precondition of action a_{i+1} and a positive effect (add) of a_i.*

We say then than an action a_0 is *relevant to the goal* if there is a causal chain that starts with a_0 and leads to the goal. We will actually refer to such complete causal chains simply as *paths*:

Definition 2 (Paths) *A path in a state s is a causal chain $a_0, p_1, a_1, p_2, a_2, \ldots p_n, a_n$ where a_n is the End action.*

This notion of relevance, considered already in (Nebel, Dimopoulos, and Koehler 1997), has a problem that is immediately apparent: a path may connect an applicable action a_0 to the goal, and yet fail to be goal-oriented in a meaningful sense.

For illustrating this and related definitions, we will appeal to a simple class of Block-World problems, where n blocks $1, 2, \ldots, n$ on the table must be arranged into a single tower with block i on block $i+1$ for all $i < n$. This so-called Tower-n domain is considered in (Vidal and Geffner 2005), where it is shown that most planners have to search in this domain in spite of its simplicity. Actually, a planner such as FF fails to solve a slight variation of the problem where block n is initially on block 1.[2]

The paths generated in the initial state of Tower-n include reasonable causal chains such as

$$t_1 : pick(i), hold(i), stack(i, i+1), on(i, i+1), End$$

for every i different than n, where a block i is picked up and stacked on its target destination, but also less reasonable ones such as:

$$t_2 : pick(i), hold(i), stack(i, j), on(i, j), unstack(i, j), \\ hold(i), on(i, i+1), End$$

where, after block i picked up, it is placed on top of another block $j \neq i+1$ along the way, for no apparent reason. Yet the irrelevant action $stack(i, j)$ is in this path for a 'reason': it supports the fluent $on(i, j)$ which is a precondition of the following $unstack(i, j)$ action that undoes the effect of the spurious action. The result of this, is that in the state s' that results from picking up block 1, any action $stack(1, i)$, and even the action $putdown(1)$, are all deemed as relevant.

A simple way to prune such spurious paths from consideration is by requiring that the actions a_i that support (add) the fluents p_{i+1} be not any supporter of p_{i+1}, but a reasonable one; e.g., a *best supporter*, which as defined above is an action a that adds p_{i+1} and has min $h_{max}(a)$. We call the resulting paths, minimal paths

Definition 3 (Minimal Paths) *A minimal path in the state s is a path a_0,p_1,a_1,p_2,a_2,...,p_n,a_n where each action a_i, for $i = 0, \ldots, n-1$, is a* best supporter *of fluent p_{i+1} in the state s.*

For example, while the path t_1 above is minimal, the path t_2 is not, as $pick(i)$ is the only best supporter of $hold(i)$ and not $unstack(i, j)$.

Minimal paths no only exclude spurious actions from getting in but have a convenient monotonicity property: if action a_i precedes a_{i+1} in the path, then $h_{max}(a_i) < h_{max}(a_{i+1})$. This follows because a_i being a best supporter of p_{i+1} implies $h_{max}(p_{i+1}) = 1 + h_{max}(a_i)$, and since p_{i+1} is a precondition of a_{i+1}, $h_{max}(p_{i+1}) \leq h_{max}(a_{i+1})$.

Minimal paths do not refer to paths with shortest length; indeed minimal paths come in different lengths. Yet if we define the *length of a path* as *the number of actions in the path*, we can establish a relation between the length of the longest minimal paths in s and the value of the h_{max} heuristic in the same state:

Proposition 4 (Path Length and h_{max} Heuristic) *The length of the longest minimal paths in s is equal to $h_{max}(s)$*

Indeed, $h_{max}(s)$ represents the length of all the minimal paths $a_0, p_1, a_1, \ldots p_n, a_n$ where p_i, for every $i = 1, \ldots, n$, represents a 'critical' precondition of the action a_i (i.e., a precondition with highest h_{max} value). The length of the minimal paths where at least one of the atoms p_i is not a critical precondition of a_i will be necessarily smaller than $h_{max}(s)$.

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 *minimal path* starting with $a_0 = a$, then we obtain that:

Proposition 5 (Helpful and Minimal Actions) *If an action a is helpful in the state s according to FF, then a is minimal in s.*

In other words, actions that are not minimal are never helpful. On the other hand, FF may declare as unhelpful actions that are minimal. This is actually because FF selects as helpful only the minimal actions that occur in the selected relaxed plan along with the minimal actions that add the precondition of an action in the relaxed plan. Minimal and helpful actions thus coincide when a single relaxed plan (taken as a set of actions) complies with the NO-OP first heuristic. Otherwise, the minimal actions correspond to the applicable actions that occur in *some* relaxed plan.

In Tower-n, there is a single relaxed plan for the initial state s_0 that includes the actions $pick(i)$ and $stack(i, i+1)$ for $i = 0, \ldots, n-1$. Thus, the minimal and helpful actions in s_0 coincide and correspond to the $pick(i)$ actions, which in turn represent the set of all the actions that are applicable in s_0. This is reasonable, as indeed, all these actions are necessary for solving the problem. Yet only the action $pick(n-1)$ makes sense in s_0; all the other pickup's do not help in s_0 as blocks i cannot be placed on top of block $i+1$ until block $i+1$ is well-placed. Heuristic search planners evaluate the heuristic $h(s_a)$ of the states that result from executing the applicable action a in s, with the hope that the action that is best leads to a state with a lower heuristic value. Yet, this doesn't always happen; in this case, actually, FF assigns the heuristic value of $2n$ to s_0 and the value $2n-1$ to **all** its possible sons s_a. FF ends up solving these problems anyway, but not due to its heuristic, that fails to distinguish the action that makes sense from all the others, but due to an extra device that finds orderings among subgoals.

We want to argue that the 'wrong' actions can be ruled out in this and many other cases, not on *heuristic* grounds – namely, because they lead to states that appear to be farther away from the goal – but on *logical* or *structural* grounds – namely, because they can be shown not to 'lead' to the goal in a precise sense that we define below.

[2] Observation due to Vincent Vidal.

Consistent Paths

They key idea is to show that certain causal chains and paths cannot be part of any 'reasonable' plan. We will articulate this idea in two parts. First we will show that side effects can be propagated along causal chains $a_0, p_1, a_1, \ldots, p_n, a_n$, provided that each segment a_i, p_{i+1}, a_{i+1} in the chain for $i < n$ is regarded as a *causal link*. Second, we will show that if an action a cannot start a causal chain ending in the goal (a path), then the action a cannot appear as the first action of any plan that is not redundant; namely, where every action supports the precondition of a later action or is the End action.

Recall that a causal link a_i, p_{i+1}, a_{i+1} states that action a_i supports the precondition p_{i+1} of action a_{i+1} and implies that no other action that either adds or deletes p_{i+1} can occur between a_i and a_{i+1}. A causal link a_i, p_{i+1}, a_{i+1} thus ensures that the fact p_{i+1} is preserved in a given interval, and yet by preserving p_{i+1} it may preserve other facts as well. For example, if q is true after the action a_i, and all actions that delete q either add or (e-)delete p as well, then due to the semantics of the causal link a_i, p_{i+1}, a_{i+1}, q must be remain true until the action a_{i+1} is applied.

We formalize the notion of side-effects along causal chains using the following definition of *labels*:

Definition 6 (Labels) *For a causal chain $t : a_o$, p_1, a_1, ..., p_n, a_n in a state s, $Label_t^-(a_i)$ and $Label_t^+(a_i)$ for $i = 0, \ldots, n$ are sets of fluents defined recursively as:*

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

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 set

$$\{q \in s \mid \forall\, a \text{ that deletes } q,\ a \text{ adds or e-deletes } p\}$$

Taking the causal chain $a_o, p_1, a_1, \ldots, p_n, a_n$ as a sequence of causal links that start in the state s, $Label_t^-(a_i)$ ($Label_t^+(a_i)$) captures what can be inferred to be true right before (resp. right after) action a_i is executed in *any plan* that makes the causal chain t true (more about this below). Thus, since a_0 is applied in s, s must be trivially true right before a_0 is applied, and $Update(a_0, s)$ must be true right after. The $Update(a_{i+1}; s)$ operator is standard as says that right after a_{i+1} whatever was true before the action a_{i+1} is applied that is not deleted, must be true after the action, along with the action positive effects. The operator $Persist(p_{i+1}, Label_t^+(a_i))$ is used in turn to infer that if q was true right after action a_i, then it will remain true at least until the action a_{i+1} is applied if every action that deletes q violates and thus is ruled out by the causal link a_i, p_{i+1}, a_{i+1}.[3]

[3]The use of the notion of e-deletes in the set $Persist(p; s)$, that appeals to the notion of mutexes, arises because a causal link that preserves a condition p does not only rule out the actions that explicitly delete p but also those that presume that p is false. On the other hand, for removing an atom q from the label, q must be explicitly deleted by an action; the actions that presume q to be false need not be taken into account.

By propagating the side effects along causal chains, we easily can detect that certain causal chains are impossible. We say then that such causal chains are *inconsistent*:

Definition 7 (Inconsistent Chains) *A causal chain a_0, p_1, a_1, ..., p_n, a_n is* inconsistent *if for some action a_i, $i = 1, \ldots, n$, $Label_t^-(a_i)$ is mutex with $Pre(a_i)$.*

Consider for example the minimal path

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

in the initial state s_0 of Tower-n. We show that this path is inconsistent for any block $k < n-1$, meaning that all such paths provide no 'evidence' that the action $pick(k)$ is relevant to the goal in s_0. For this, we show that the atom $ontable(k+1)$ which is true in s_0, gets propagated until the end of the path, thus being part of $Label_t^-(End)$. Then since $ontable(k+1)$ for $k < n-1$ is mutex with the real goal $on(k+1, k+2)$, which is a precondition of the action End, the path t must be inconsistent. Notice that $ontable(k+1)$ is true right after the action $pick(k)$ in s_0. Moreover, the only action that deletes $ontable(k+1)$ is $pick(k+1)$ which e-deletes the condition $hold(k)$ in the first causal link of t (this is because $hold(k)$ is mutex with the precondition $armfree$ of $pick(k+1)$ and is not added by the action). This means that $ontable(k+1)$ must be true right before, and also right after the action $stack(k, k+1)$ in t. Finally, since the only action that deletes $ontable(k+1)$ is $pick(k+1)$ which also e-deletes $on(k, k+1)$ (the action has a precondition $clear(k+1)$ that is mutex with the atom $on(k, k+1)$ that does not add), then we obtain the atom $ontable(k, k+1)$ must be true right before the action End. Indeed, the labels that are obtained for the path t above for any block $k < n-1$, that we abbreviate as $L_t^-(a_i)$ and $L_t^+(a_i)$ for each of the 3 actions a_i in t (pick,stack,End) are:

- $L_t^-(a_1) = \{ontable(j), clear(j), armfree\}$
- $L_t^+(a_1) = \{ontable(i), clear(i), hold(k)\}$
- $L_t^-(a_2) = \{ontable(i), hold(k)\}$
- $L_t^+(a_2) = \{ontable(i), on(k, k+1), clear(k), armfree\}$
- $L_t^-(a_3) = \{on(k, k+1), clear(k), ontable(k, k+1)\}$

where i and j range over $[1..n]$ with i being different than k.

Before describing the formal properties of consistent paths, let us define the *consistent actions* in s as follows:

Definition 8 (Consistent Actions) *An action a is consistent in s iff it is the first action in a consistent path in s.*

Formal Properties

There are good and bads news about consistent paths: the good news is that inconsistent paths and actions can be safely ignored; the bad news is that determining whether there is a consistent path is computationally intractable. Our strategy will be to keep the good part and avoid the bad part by considering later the *minimal* paths that are consistent only.

Soundness

Causal links have been used in the context of partial order planning, where branching is carried out by selecting flaws in the plan (open supports and causal link threats) and trying out the possible repairs. We can interpret the semantics of causal chains and paths in the context of partial order planning, but fortunately we do not need to. Since we aim to use these notions in the context of a forward-state planner, we can rather talk about the implicit *causal structure* of standard, sequential Strips plans, and state the conditions under which a causal chain is true in a plan. For this, we need to map actions in the chain to actions in the plan, as the same action can appear multiple times in the plan. We use the notation $\pi(k)$ to refer to the k-th action in the plan π.

Definition 9 *A causal chain a_0, p_1, a_1, ..., p_n, a_n is true in a plan $\pi = b_0, \ldots, b_m$ for the mapping $f(i)$, $i = 0, \ldots, n$, $0 < f(i) < f(i+1) \leq m$, if 1) $a_i = \pi(f(i))$ for $i = 0, \ldots, n$, 2) $f(0) = 0$, and 3) $\pi(f(i))$ adds the precondition p_{i+1} of $\pi(f(i+1)$ for $i = 0, \ldots, n-1$ and no action b_k in the plan , $i < k < j$, adds or deletes p.*

The definition says that the actions in the chain occur in the right order in the plan, that the first action in the chain and in the plan coincide, and that the causal links in the chain are all true in the plan. The side effects captured in the $Label$ function can then be shown to be sound in the following sense:

Proposition 10 *If the causal chain $t : a_0$, p_1, a_1, ..., p_n, a_n is is true in the plan $\pi = b_0, \ldots, b_m$ for the mapping $f(i)$, then the atoms in $Label_t^-(a_i)$ ($Label^+(a_i)$) are true in the plan right before (resp. right after) the action $\pi(f(i))$ is executed in the plan π .*

Proposition 11 *If the path $a_0, p_1, a_1, \ldots, p_n, a_n$ is true in some plan π for some mapping, then the path must be consistent.*

This means that inconsistent paths cannot be true in any plan for any mapping. Let us say then that a plan $\pi = b_0, \ldots, b_m$ is *irredundant* if every action b_i, $i = \ldots, m$, $b_i \neq End$, is the support of the precondition p of an action b_j, $0 < i < j \leq m$ (i.e., there is a true causal link b_i, p, b_j in π for every action b_i in π different than the End action). Then

Proposition 12 *If the action a is inconsistent in s, then a cannot be the first action of any irredundant plan from s.*

Summarizing the results so far: a path $t : a_0$, p_1, a_1, ..., p_n, a_n can be taken as evidence of the relevance of the action a_0 to the goal except when the path t is inconsistent, and hence impossible in any (irredundant) plan. Inconsistency pruning is thus *sound* but it is *not complete*; i.e., an action a may be consistent in s and yet not head any (irredundant) plan. Indeed, later on we will consider another version of path consistency that is stronger, sound, and polynomial, but hence incomplete as well.

Complexity

While determining whether a given path is consistent can be done in polynomial time, the problem of determining whether there is a consistent path is intractable.

Proposition 13 *Determining whether there is a consistent path from a state s is NP-Complete*

The result follows from a reduction from the NP-Complete problem 'Paths with Forbidden Pairs' (PFP): this is the problem of determining whether there is a path in a directed graph from a given source node n_0 to a given target node n_m that contains at most one node from each pair of nodes in a 'forbidden pairs list' (Garey and Johnson 1979). It suffices to create for each node n_i two fluents p_i and q_i, and for each directed edge $n_i \rightarrow n_j$ in the graph, an action with preconditions p_i and q_i, positive effect p_j, and negative effects q_k for each node n_k in a forbidden pair (n_j, n_k). In the initial state all fluents q_i must be true for all i along with the fluent p_0. The goal is p_m. There is then a 1-to-1 correspondence between the causal chains that terminate in the goal and the directed paths in the graph that connect n_0 to n_m (no action adds fluents q_i, hence all causal chains can mention only the fluents p_i), and also a 1-to-1 correspondence between the directed paths connecting n_0 and n_m with no forbidden pairs and the consistent causal chains ending in the goal.

Minimal Consistent Paths: Computation

We seem to have arrived nowhere: the computation of consistent paths is not only computationally hard, but it turns out to be pretty useless too; few actions can be pruned in this way. Yet, a simple refinement will solve both of these problems: rather than considering the paths that are consistent, we will consider only the *minimal paths* that are so. We call these the *minimal consistent paths*: these are the paths $a_0, p_1, a_1, \ldots, p_n, a_n$ that are consistent and where each action a_i is a *best supporter* of fluent p_{i+1} (actions a that add p_{i+1} and have min $h_{max}(a)$).

Rather than pruning an action a in state s when it is not the first action in a consistent path, we will prune it when it is not the first action in a *minimal consistent path*. The resulting pruning criterion, that we refer to as *minimal consistency*, is neither sound nor complete, but it seems adequate from an heuristic point of view: it tends to be correct (pruning actions that are not good), powerful (pruning many actions), and efficient (as it can be computed sufficiently fast, even if it is also intractable in the worst case).

We focus now on the computation of all the minimal consistent paths. The algorithm computes the labels $Label_t^-(a_i)$ and $Label_t^+(a_i)$, for all *minimal paths* t, while pruning the the minimal paths that are not consistent. Recall that a path $t = a_0, p_1, a_1, \ldots, p_n, a_n$ is not consistent if for some action a_i in the path, $Label_t^-(a_i)$ is mutex with $Pre(a_i)$.

While $Label_t^-(a_i)$ and $Label_t^+(a_i)$ were used before to carry the side effects that are true before and after the action a_i, $LBS^-(a_i)$ and $LBS^+(a_i)$ below will capture the *collection of labels* that express the true conditions before and after a_i in a plan that complies with *some minimal consistent causal chain up to a_i*. In other words, L will be in $LBS^-(a_i)$ ($LBS^+(a_i)$) if and only if $L = Label_t^-(a_i)$ (resp. $L = Label_t^+(a_i)$) for some consistent minimal causal chain $t : a_0, p_1, \ldots, a_i$.

We use some abbreviations in the algorithm: $O^*(p)$ stands for the best supporters of fluent p, $P(p_{i+1}; s')$ refers to $Persist(p_{i+1}; s')$, and actions a_i stand for actions in minimal paths with $h_{max}(a_i) = i$. The algorithm then proceeds in four steps:

1. Compute h_{max} for all p and a
2. Determine the minimal paths
3. Propagate labels along minimal paths keeping pointers

$LBS^-(a_0) = \{s\}$
$LBS^+(a_i) = \{Update(a_i; s') \mid s' \in LBS^-(a_i)\}$
$LBS^-(a_{i+1}) = \{P(p_{i+1}; s') | p_{i+1} \in Pre(a_{i+1}), s' \in LBS^+(a_j),$
$a_j \in O^*(p_{i+1})$ s.t. $Pre(a_{i+1})$ **not mutex with** $P(p, s')\}$

4. Read off the consistent minimal paths

Step 1 and 3 involve forward passes from the initial situation: the first to compute the heuristic h_{max} and the third to propagate the labels. Steps 2 and 4, on the other hand, involve backward passes from the goal (End action): the second to determine the minimal paths, the fourth, to determine the minimal paths that are consistent. Step 3 takes advantage that the propagation of labels along a given path is markovian (only the last label matters), and prunes a path as soon it is found to be inconsistent (this is the role of the mutex condition). The consistent minimal paths are retrieved in Step 4 following stored pointers; namely, using the precondition p_{i+1}, label s', action a_j responsible for the inclusion of a label in $LBS^-(a_{i+1})$, Step 3.

In the worst case, this computation grows with the number of minimal paths, which can be exponential although seldom is. A tigher measure of the complexity of the computation is obtained by assessing the size of the collections of labels $|LBS^-(a_i)|$ and $|LBS^+(a_i)|$ that measure the number of minimal paths up to the action a_i that are consistent, which in most of the benchmarks appears to be small.

The most basic planner that we will consider is the C1 planner that is a simple forward-state backtracking planner that repeats this computation in every state visited, pruning the actions that do not head any consistent path (the inconsistent actions). We will see that this planner, while simple, is very effective, solving more problems than the basic architecture of FF captured by the EHC search, taking only 20% more time. From a pool of 546 problems, FF(EHC) solves 78% while C1 solves 80%, more than half of them backtrack-free. The results appear in the columns for FF(EHC) and C1 of Table 1 below.

Extensions

We consider two extensions of this basic C1 planner. One extension is aimed at making the consistency path criterion stronger, the other is aimed at exploiting further the applicable causal links that are found to be consistent.

Consistency: Negation and Joint Persistence

Two simple ways for strengthening the notion of path consistency while keeping it sound result from using *negative literals* in labels in addition to atoms, and a stronger function $Persist(p; s)$ for capturing the literals in s that must persist when the condition p is preserved.

Negative literals can be introduced in the labels of chains $t = a_0, p_1, a_1, \ldots, p_n, a_n$ through a slight change in the definition of the $Label$ function above (the change in the LBS functions is then direct).[4] First, negative literals $\neg L$ that are true in the seed state s are added to $Label_t^-(a_0)$. Second, the $Update(a; s)$ function is revised so that negative literals $\neg L$ in s are removed when a adds L, while negative literals $\neg L$ are added when a deletes L. And third, $Persists(p; s)$ is revised as well, so that negative literals $\neg L$ in s are allowed to persist when all actions a that add L, either add or e-delete p. Once negation is added, two complementary literals must be taken as mutex too.

For a stronger definition of the function $Persists(p; s)$, we take *the maximal subset* $Q \subseteq s$ such that for every $q \in Q$, all the actions a that delete q, add or e-delete p, or have a precondition that is mutex with an atom $q' \in Q$. It is straighforward to extend this stronger definition to both positive and negative literals. The set Q can then be computed iteratively, initializing Q to s, and removing from Q every atom q that violates the above condition until no more such atoms are left in Q. The computation of this stronger form of *joint persistence* is more costly but in many cases, pays off.

Committing to Causal Links

An action a_0 with effects p_1 and p_1' can be consistent in a state s because it heads a path $a_0, p_1, a_1, \ldots, p_n, a_n$ that is consistent, even if there is no consistent path $a_0, p_1', a_1', \ldots, p_m, a_m$ for the second effect p_1'. The path consistency criterion thus selects *causal links* rather than actions. In this case, it says that the causal link a_0, p_1, a_1 is consistent in s as it can be extended into a complete causal chain, while the causal link a_0, p_1', a_1' is not.

This suggests that when an action a is selected in a state s for application because there is a causal link a, p, b that is consistent in s, aside from applying the action a to s resulting in the new state s_a, we may also *commit* to the causal link a, p, b by extending the *search state* in the planner to reflect that the condition p, true in the state s_a, is to be maintained *excluding* the application of actions that add or delete p until the action b is done.

We take this idea further. Given a search state that contains a set CLs of commitments a, p, b, we *enforce* the committed causal links in the computation of the heuristic h_{max} that determines the best supporters $O^*(p)$ and the minimal paths. For this, every action c that adds or e-deletes p, must be 'pushed' over the action b, by enforcing the constraint $h_{max}(c) > h_{max}(b)$. This is because any such action must either come before a or after b, but since a has already been executed, c must come after b. Enforcing the constraint $h_{max}(c) > h_{max}(b)$ in the computation of the h_{max} heuristic is straightfoward, as it basically involves taking a (non-

[4] Alternatively, the same definition of labels can be used, but assuming that atoms $\bar{p}$ representing the negation of the atoms $p \in F$ in the problem have been added, as when negation is compiled away (Gazen and Knoblock 1997). The result is equivalent.

		FF(EHC)		C1				C2+			
Domain	I	S	T	S	BF	NS	T	S	BF	NS	T
Blocks World	50	42	0.22	46	26	4M	1.63	50	49	–	2.13
Depots	22	19	44.08	18	11	4T	36.82	18	14	4T	46.25
Driver	20	6	41.28	12	8	8T	21.65	15	7	5T	267.97
Ferry	50	50	$<$ 1 msec	50	6	–	0.03	50	50	–	0.09
Free Cell	20	14	39.88	2	2	18M	6.58	3	1	17T	397.06
Grid	5	5	$<$ 1 msec	3	–	1T,1M	186.26	4	1	1T	438.86
Gripper	50	50	$<$ 1 msec	50	–	–	0.16	50	50	–	1.45
Logistics	28	28	$<$ 1msec	28	28	–	0.11	28	28	–	0.33
Miconic	50	50	$<$ 1 msec	50	50	–	0.01	50	50	–	0.03
Mystery	30	15	323.08	20	8	1R,9T	4.83	24	22	2R,4T	4.73
Open Stacks	30	30	7.12	28	28	2T	381.04	7	3	23T	654.88
Pipes World	50	4	18.01	10	4	9T,31M	150.32	12	5	28T,10M	89.05
Rovers	40	40	26.97	36	36	4T	448.96	32	31	8T	22.91
Satellite	20	20	0.02	20	20	–	2.13	20	20	–	1.73
Storage	30	3	15.00	27	11	2T,1M	150.28	12	6	18T	99.56
TPP	30	30	426.90	16	0	3T,11M	170.01	11	8	9T,10M	36.7
Zeno Travel	20	18	2.04	20	19	–	79.25	18	18	2T	12.64
Total	546	425	78.72	436	257		96.47	404	363		122.14
Percentage	100%	78%		80%	47%			74%	66%		

Table 1: Coverage of FF-EHC, C1, and C2+: S is the total number of solved instances, NS stands for the unsolved instances with nT, nM, and nR meaning the number of time outs, memory outs, and no solutions found after finishing the (incomplete) search. T is average time in seconds and BF stands for the number of instances solved backtrack-free.

existing) dummy effect of b as a precondition of c. Two effects of committing to and enforcing causal links in this way are that actions c that violate a committed causal link will have heuristic values $h_{max}(c) > 0$, and more importantly, paths that were not minimal before the commitments may become minimal after the commitments.

Planners

The C1 planner, discussed above, is a forward-state backtracking planner that simply prunes inconsistent actions, using the original definition of consistency. The C2 planner incorporates two extensions over C1: the stronger consistency criterion based on the use of negation and joint persistence, and the commitments to the causal links enforced in the computation of the h_{max} heuristic. These are both pure planners that provide a basic bottom-line. They do surprisingly well given that they are based on purely structural selection criteria; the h_{max} heuristic capturing a basic reachability notion and not being a useful heuristic in itself. The planner C2, however, is not a clear improvement over C1: the reason is that while it uses stronger forms of inference, it multiplies the branching factor (actions a vs. causal links a, p, b) and adds commitments that make the rest of the search more difficult. In order to obtain a better performance out of the basic C2 architecture we consider two simple improvements on C2. First, rather than committing to a single causal link a, p, b when an action a is done, we establish a commitment to a *disjunction* of causal links a, p, b_1; a, p, b_2; $\ldots$; a, p, b_m that differ only in the target action b_i for links a, p, b_i that head a consistent minimal path. These 'disjunctive' causal links are removed from the set of commitments when *any* action b_i is done. Also, the actions c that violate this link are pushed to come after *any* of these actions, the constraint enforced being $h_{max}(c) > \min_i h_{max}(b_i)$. The second change in C2 is that the consistent causal links a, p, b are ordered by the h_{max} value of the search state that would result from applying the action a and adding the commitment a, p, b. We refer to the improved C2 planner as C2+.

Experimental Results

We evaluate the planners C1 and C2+ in relation to FF using only EHC, which is the incomplete planner that constitutes the basic architecture of FF and accounts for its speed. In FF, a complete but slower best-first search (BFS) mode is triggered when EHC fails. We do not compare with FF running in both modes, as one could also switch from C1 or C2+ to such BFS when they fail after a few minutes or simply when they have to backtrack. As we will see, C2+ solves without backtracks 85% of the 425 problems that are solved by FF(EHC) with times and plans that are not very different than FF. C1 and C2+ are written in C++ and use the parser from Metric-FF. The experiments below were conducted on a CPU with a clock speed of 2.33 GHz and 8 Gb of RAM with cutoffs in time and memory of 2 hours and 2Gb.

Table 1 shows the coverage of the three planners on a collection of 546 benchmarks, many of them used in previous Int. Planning Competitions. FF(EHC) solves 78% of the problems while C1 and C2+ solve 80% and 74% problems each, 47% and 66% backtrack-free respectively. In terms of overall time, C1 and C2+ trail FF(EHC) but are not that behind. FF(EHC) does definitely best in Free-Cell, Rovers, Grid, and TPP, while C1 and C2+ do best in Blocks, Driver, Mystery, PipesWorld, and Storage. C2+ does not improve C1 in the number of problems solved, but improves C1 in the number of problems solved backtrack-free (66% vs. 47%) and plan quality.

Table 2 shows the quality of the plans found by C1, C2+, and the problems solved backtrack-free by C2+ in relation to FF. The plans found by C1 tend to be very long, which is not surprising, as C1 does not use any criterion for ordering the consistent actions. C2+ does better, in particular in the problems that it solves backtrack-free, where the lengths of the plans are slightly above the lengths of the plans found by FF. The table, however, does not include two 'outliers': the domain Grid and TPP, where C1 finds plans that are 22177% and 3831% worse than FF. C2+ finds also crazy plans in these domains which are 1342% and 643% worse. On the other hand, on the instances that C2+ solves backtrack-free, the first number is 349% and the second is 103%.

Domain	C^1	C^{2+} (All)	C^{2+} (BF only)
Blocks World	564%	131%	132%
Depots	848%	108%	98%
Driver	104%	101%	101%
Ferry	149%	116%	116%
Free Cell	285%	138%	122%
Gripper	132%	127%	127%
Logistics	195%	116%	116%
Miconic	151%	101%	101%
Mystery	265%	160%	139%
Open Stacks	98%	95%	88%
Pipes World	190%	113%	124%
Rovers	114%	110%	109%
Satellite	114%	103%	103%
Storage	144%	100%	100%
Zeno Travel	274%	138%	138%
Average	242%	130%	113%

Table 2: Plan Quality of C1,C2+, and C2+ (BF): Entries express number of actions in plan found vs. number of actions in FF's plan; e.g., 200% means plans on average twice as long as in FF.

Discussion

We have approached the problem of inference in planning from a different perspective, building on the notion of *causal chains;* sequences of causal links. We have shown that the semantics of causal links along such chains can be used to detect that some of the chains are inconsistent and cannot be true in any plan, and furthermore, that actions that do not head any consistent causal chain leading to the goal can be pruned. We showed that while simple, this inference rule is quite powerful, as a plain backtracking forward-state search planner using with a version of this rule, that considers only the *minimal paths,* solves as many benchmark problems as the effective Enforced Hill Climbing search of FF, many of them backtrack-free. We also considered some extensions of this basic architecture, that result from improving the definition of consistent paths, and adding causal links commitments during the forward-state search.

Most of the information that is captured by the computation of the consistent minimal paths is thrown away in each node, except for the first action or link. Making further use of this information remains a challenge that we hope to address in the future. At the same time, we hope that this perspective opens new doors to the problem of inference in planning that could lead to improvements in current planning technology on the one hand, and to a better comprehension of the types of planning problems that are simple, on the other. It appears, for example, that in simple problems, there are always minimal consistent paths that provide a route to the goal, while in puzzle-like problems, the minimal paths are often inconsistent. Interestingly, the EHC search in FF relies also on paths that are minimal, and hence, the range of problems that are solved by (minimal) consistent pruning and by FF's EHC appear to be closely related, even if in the former there is no explicit heuristic measure of progress. Nonetheless, finding out if there is a measure of progress that is improved when consistent actions are applied, is an interesting topic for future research as well.

References

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

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

Chen, Y.; Wah, B. W.; and C.Hsu. 2006. Temporal planning using subgoal partitioning and resolution in sgplan. *J. Artif. Intell. Res. (JAIR)* 26:323–369.

Garey, M., and Johnson, D. 1979. *Computers and Intractability*. Freeman.

Gazen, B., and Knoblock, C. 1997. Combining the expressiveness of UCPOP with the efficiency of Graphplan. *Lect. Notes in AI 1348*, 221–233. Springer.

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

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

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

McDermott, D. 1999. Using regression-match graphs to control search in planning. *AI* 109(1-2):111–159.

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*.

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*. Springer.

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

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