Searching for Plans with Carefully Designed Probes

Nir Lipovetzky
DTIC Universitat Pompeu Fabra
Barcelona, SPAIN
nir.lipovetzky@upf.edu

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

Abstract

We define a probe to be a single action sequence computed greedily from a given state that either terminates in the goal or fails. We show that by designing these probes carefully using a number of existing and new polynomial techniques such as helpful actions, landmarks, commitments, and consistent subgoals, a single probe from the initial state solves by itself 683 out of 980 problems from previous IPCs, a number that compares well with the 627 problems solved by FF in EHC mode, with similar times and plan lengths. We also show that by launching one probe from each expanded state in a standard greedy best first search informed by the additive heuristic, the number of problems solved jumps to 900 (92%), as opposed to FF that solves 827 problems (84%), and LAMA that solves 879 (89%). The success of probes suggests that many domains can be solved easily once a suitable serialization of the landmarks is found, an observation that may open new connections between recent work in planning and more classical work concerning goal serialization and problem decomposition in planning and search.

Introduction

Heuristic search has been the mainstream approach in planning for more than a decade, with planners such as FF, FD, and LAMA being able to solve problems with hundreds of actions and variables in a few seconds (Hoffmann and Nebel 2001; Helmert 2006; Richter and Westphal 2010). The basic idea behind these planners is to search for plans using a search algorithm guided by heuristic estimators derived automatically from the problem (McDermott 1996; Bonet and Geffner 2001). State-of-the-art planners, however, go well beyond this idea, adding a number of techniques that are specific to planning. These techniques, such as helpful actions and landmarks (Hoffmann and Nebel 2001; Hoffmann, Porteous, and Sebastia 2004; Richter, Helmert, and Westphal 2008), are designed to exploit the *propositional structure* of planning problems; a structure that is absent in traditional heuristic search where states and heuristic evaluations are used as *black boxes*. Moreover, new search algorithms have been devised to make use of these techniques. FF, for example, triggers a best-first search when an incomplete but effective greedy search (enforced hill climbing) that uses helpful actions only, fails. In FD and LAMA, the use of helpful or preferred operators is not restricted to the first phase of the search, but to one of the open lists maintained in a multi-queue search algorithm. In both cases, dual search architectures that appeal either to two successive searches or to a single search with multiple open lists, are aimed at quickly solving, large problems that are simple, without giving up completeness on problems that are not.

In this work, we formulate and test a new dual search architecture for planning that is based on the idea of *probes*: single action sequences computed without search from a given state that can quickly go deep into the state space, terminating either in the goal or in failure. We show that by designing these probes carefully using a number of existing and new polynomial inference techniques, 683 out of 980 (70%) problems from previous IPCs can be solved with a single probe from the initial state. Moreover, by using one probe as a lookahead mechanism from each expanded state in a standard greedy best first search informed by the additive heuristic, the number of problems solved jumps to 900 (92%), a number that compares well to state-of-the-art planners like FF and LAMA that solve 827 (84%) and 879 (89%) problems respectively.

The main contribution of the paper is the design of these probes. A probe is an action sequence computed greedily from a seed state for achieving a *serialization of the problem subgoals* that is computed dynamically along with the probe. The next subgoal to achieve in a probe is chosen among the first unachieved landmarks that are *consistent*. Roughly, a subgoal that must remain true until another subgoal is achieved, is consistent, if once it is made true, it does not have to be undone in order to make the second subgoal achievable. The action sequence to achieve the next subgoal uses standard heuristics and helpful actions, while maintaining and enforcing the reasons for which the previous actions have been selected in the form of *commitments* akin to causal links. The computational value of the subgoal serialization, the consistency checks, and the use of commitments, is evaluated empirically as well.

The use of lookahead in search and planning is very old in AI, and appears more recently in the YAHSP planner that makes an attempt to look ahead by using sequences of actions extracted from the relaxed plan (Vidal 2004). While PROBE also looks ahead by using sequences of actions, the design and use of these sequences is completely different in the two planners. In particular, while in YAHSP, the action sequences are executable prefixes of the relaxed plan, in PROBE, they are computed from scratch to achieve each one of the remaining subgoals in sequence. The range of domains that are solved by just throwing a single probe from the initial state is much larger. In this sense, the motivation for PROBE is related to the motivation behind other recent planners such as eCPT (Vidal and Geffner 2005) and C3 (Lipovetzky and Geffner 2009) that also aim to solve simple, non-puzzle-like domains, with little or no search at all. This requires capturing in a domain-independent form the inferences that render the search superfluous in such domains. This task is non-trivial, but as shown here, it can pay off even in planners that do search.

The idea of searching with probes has been considered before in the form of *random probes* (Langley 1992). *Limited discrepancy search* can be thought of as a systematic method for searching with probes (Harvey and Ginsberg 1995), while *Monte Carlo planning*, as a non-systematic method that uses multiple random probes (Nakhost and Müller 2009). PROBE, in contrast, uses single, carefully designed probes. Below, we present the planner first, the experimental results, and then some conclusions.

PROBE: The Planner

Heuristic search planners that just plug a delete-relaxation heuristic into a well known search algorithm are nice, as they can be easily understood. A problem that they face, however, are the search plateaus, a situation that arises when goals are in 'conflict', and approaching one means to move away from the others. Since the formulation of *more effective estimators* hasn't been simple after more than a decade, the solution to this problem has given rise to other types of inferences and techniques. These techniques are absent in the first generation of planners such as UNPOP and HSP, but are present in FF, FD, and LAMA. These planners are less monolithic, and their details are often more difficult to follow, but it's precisely those 'details' that make the difference. The planner PROBE is no exception to this trend towards 'finer-grained planning', and incorporates a number of design decisions that we explain below.

PROBE is a complete, standard greedy-best first (GBFS) STRIPS planner using the standard additive heuristic, with just *one change*: when a state is selected for expansion, it first launches a *probe* from the state to the goal. If the probe reaches the goal, the problem is solved and the solution is returned. Otherwise, the states expanded by the probe are added to the open list, and control returns to the GBFS loop. *The crucial and only novel part in the planning algorithm is the definition and computation of the probes.*

We assume a STRIPS problem whose top goals G are the preconditions of a dummy End action that adds a dummy goal G_d. As in POCL planning, this is needed due to the use of causal commitments that are similar to causal links (Tate 1977; McAllester and Rosenblitt 1991).

Probes

A *probe* is an action sequence $a_0, a_1, \ldots, a_k$ that generates a sequence $n_0, n_1, \ldots, n_{k+1}$ of nodes, each of which is a pair $n_i = \langle s_i, C_i \rangle$ made up of the problem state s_i and a set of *causal commitments* C_i. The initial node of a probe is $n_0 = \langle s, \emptyset \rangle$ where s is the state from which the probe is triggered, and $\emptyset$ is the empty set of commitments. The *action selection* criterion decides the action a_i to choose in node $n_i = \langle s_i, C_i \rangle$ greedily without search. This action generates the new node to $n_{i+1} = \langle s_{i+1}, C_{i+1} \rangle$, where s_{i+1} is the result of progressing the state s_i through a_i, and C_{i+1} is C_i updated with the causal commitments *consumed* by a_i removed, and the causal commitments *produced* by a_i added.

Probe Construction

The actions in a probe are selected in order to achieve *subgoals* chosen from *the landmarks that are yet to be achieved.* A number of techniques are used to make the greedy selection of the *next subgoal* to achieve and *the actions for achieving it*, effective. A probe that reaches the goal is the composition of the action sequences selected to achieve the next subgoal, the one following it, and so on, until all landmarks including the dummy goal G_d are achieved. Probes are not and need not be complete; yet they are supposed to capture the plans that characterize 'simple domains' even if a formal characterization of such domains is still missing.

The subgoal to pursue next is selected in a node n in two cases: when n is the first node of the probe, or when the subgoal g associated with its parent node n' in the probe is achieved in n. Otherwise, n *inherits the subgoal from its parent node*. The action a selected in a node n is then the action that appears to be 'best' for the subgoal g associated with n. If a does not achieve g, then g stays active for the next node, where the action to include in the probe is selected in the same way.

The formal definition of the subgoal and action selection criteria below uses notions that will be made fully precise later on, like the *heuristic* $h(G|s, C)$ that takes both the state s and the *commitments* C into account, the precomputed *partial ordering among landmarks*, and the conditions under which a subgoal is deemed as *consistent* from a given node.

Subgoal and Action Selection

The criterion for selecting the subgoal g in node $n = \langle s, C \rangle$ is the following. First, the set S of *first unachieved landmarks* that are *consistent* in $n = \langle s, C \rangle$ is computed. Then, the landmark $p \in S$ that is *nearest* according to the heuristic $h(p|s, C)$ is selected as the subgoal for n.

The selection of the action a in n is in turn the following. First, the set of actions a that are deemed *helpful* in $n = \langle s, C \rangle$ for either the subgoal or commitments associated with n are computed, and those that lead to a node $n' = \langle s', C' \rangle$ for which either $h(G|s', C')$ is infinity or s' has been already generated are pruned.[1] Then, among the remaining actions, if any, the action that minimizes the heuris-

[1]Notice that we are forcing probes to explore new states only.

tic $h(g|s', C')$ is selected.[2] In case of ties, two other criteria are used lexicographically: first 'min $\sum_L h(L|s', C')$', where L ranges over the first unachieved landmarks, then 'min $h(G_d|s', C')$', where G_d is the dummy goal.

If a node $n = \langle s, C \rangle$ is reached such that all helpful actions are pruned, a second attempt to extend the current probe is made before giving up. PROBE recomputes the relaxed plan from n with those actions excluded, resulting in a new set of helpful actions if the heuristic does not become infinite. The new set of helpful actions is pruned again as above, and the process is iterated, until a non-pruned helpful action is obtained at s, or the heuristic becomes infinite. In the latter case, the probe *terminates* with *failure*. If before failing, it reaches a goal state, it terminates *successfully* with the problem solved.

In the next few sections, we fully specify the notions assumed in these definitions.

Commitments and Heuristic

A *causal commitment* is a triple $\langle a, p, B \rangle$ where a is an action, p is a fluent added by a, and B is a *set of fluents*. The intuition is that fluent p was added by a in order to achieve (at least) *one* of the fluents in B, and hence that p should remain true until an action adds some fluent in B, *consuming* the causal commitment. A result of this is that in a node $n = \langle s, C \rangle$ with a commitment $\langle a, p, B \rangle$ in C, any action a applicable in s that deletes p but does not add any fluent in B, is taken to *violate* the commitments in C, and is pruned from the set of applicable actions.

A heuristic $h(G|s, C)$ is used to estimate the cost to a set G of fluents from a node $n = \langle s, C \rangle$. This heuristic takes the set of causal commitments C into account and is defined like the standard additive heuristic:

$$h(G|s, C) = \sum_{p \in G} h(p|s, C) \tag{1}$$

where

$$h(p|s, C) = \begin{cases} 0 & \text{if } p \in s \\ \min_{a \in O(p)} [cost(a) + h(a|s, C)] & \text{otherwise} \end{cases} \tag{2}$$

and

$$h(a|s, C) = \delta(a, s, C) + h(Pre(a)|s, C) \ . \tag{3}$$

where $O(p)$ stands for the actions adding p and $Pre(a)$ for the preconditions of a.

The only novelty in this definition is the offset term $\delta(a, s, C)$ that penalizes actions a that violate causal commitments $\langle a_i, p_i, B_i \rangle$ in C. The offset for such actions is the cost of achieving one of the fluents in B_i, as the action a cannot be executed until those commitments are consumed. More precisely:

$$\delta(a, s, C) = \begin{cases} 0 & \text{if } a \text{ violates no commitment in } C \\ \max_i \min_{q \in B_i} h(q|s, C) & \text{otherwise,} \end{cases}$$

where B_i are the sets of fluents in the commitments $\langle a_i, p_i, B_i \rangle$ in C violated by a.

The result of the offsets arising from the commitments C is that actions a *applicable* in s may get a heuristic value $h(a|s, C)$ greater than zero when they violate a commitment in C. Likewise, a goal G reachable from s may get an infinite heuristic value $h(G|s, C)$, as for example when G requires an action a with an infinite offset $\delta(a, s, C)$. This can happen when in order to consume any of the commitments $\langle a_i, p_i, B_i \rangle$ in C violated by a, it is necessary to violate one of such commitments. For instance, if the goal G stands for the atoms $on(1, 2)$ and $on(2, 3)$ in Blocks, the heuristic $h(G_d|s, C)$ associated with the node $n = \langle s, C \rangle$ that results from stacking 1 on 2 when 2 is not on 3, will have infinite value. The reason is that the offset $\delta(a, s, C)$ for the required action $a = unstack(1, 2)$ is infinite, as a violates the commitment $\langle stack(1, 2), on(1, 2), \{G_d\} \rangle$ in C, which cannot be consumed from the state s by any other action, as G_d cannot be achieved without undoing first $on(1, 2)$.

The *relaxed plan* associated with a node $n = \langle s, C \rangle$ and a goal G is obtained by collecting backwards from G, the best supporters a_p for each p in G, and recursively the best supporters for their preconditions that are not true in s (Keyder and Geffner 2008). The best supporter for an atom p is an action a that adds p and has minimum $h(a|s, C)$ value. The *helpful actions* for a subgoal g in a node $n = \langle s, C \rangle$ are defined as in FF, as the actions a with heuristic $h(a|s, C) = 0$ that add a precondition or goal in the relaxed plan. For convenience, however, this relaxed plan is not defined as the relaxed plan for g in n, but as the relaxed plan for the joint goal formed by g and *the (disjunctive) targets* B_i in the commitments $\langle a_i, p_i, B_i \rangle$ in C. This reflects that such targets also represent subgoals associated with the node $n = \langle s, C \rangle$, even if unlike g, they do not have to be achieved necessarily.[3]

An action a selected in a node $n = \langle s, C \rangle$ generates the new node $n' = \langle s', C' \rangle$ where s' is the result of progressing s through a, and C' is the result of removing the commitments consumed by a in n, and adding the commitments made by a in n. The action a *consumes* a commitment $\langle a_i, p_i, B_i \rangle$ in C if a adds a fluent in B_i (whether or not a deletes p_i). Likewise, a *makes* the commitments $\langle a, p, B \rangle$ in $n = \langle s, C \rangle$, if p is a fluent added by a, and B is the set of fluents added by actions in the relaxed plan in n that have p as a precondition.

Disjunctive Commitments

For the purpose of the presentation, we have made a simplification that we now correct. From the description above, it's clear that an action a can introduce commitments $\langle a, p_i, B_i \rangle$ for more than one effect p_i of a. This will be the case when

This is an heuristic decision that does not compromise the completeness of the best-first search algorithm that uses probes.

[2]Except for a few details, this criterion is similar to the one used by LAMA for preferring actions in the landmark heuristic queue; namely, that "if no acceptable landmark can be achieved within one step, the preferred operators are those which occur in a relaxed plan to the nearest simple acceptable landmark" (Richter and Westphal 2010).

[3]Indeed, a probe may reach the goal with a non-empty set of commitments.

the preconditions of the actions in the relaxed plan involve more than one effect of a. The heuristic $h(G|s, C)$ and the notions above are all correct provided that this situation doesn't arise. On the other hand, when it does, the above definitions interpret multiple commitments $\langle a, p_i, B_i\rangle$ in C for a common action a *conjunctively*, as if each such commitment must be respected. This, however, is too restrictive. If a adds two relevant effects p_1 and p_2, this rules out the possibility that a is the causal support of p_1 in the plan but not of p_2. This happens for example when a block A must be placed on top of block C, given that A is on B, and B on C. In such a case, the action $pickup(A, B)$ is done in order to get the precondition $clear(B)$ of $pickup(B, C)$, but not for getting the precondition $hold(A)$ of $stack(A, C)$. Thus, in PROBE, multiple commitments $\langle a, p_i, B_i\rangle$ for the same action a in C are treated not *conjunctively*, but *disjunctively*. Namely, it's assumed that every action in a probe is made with *some* purpose encoded by a commitment, but not with all purposes that are possible. This means three things. First, an action a in a node $n = \langle s, C\rangle$ will be taken to violate a *disjunctive* commitment $\langle b, p_i, B_i\rangle$, $i = 1, \ldots, n_b$, when these are all the commitments involving the action b in C, and a violates each one of them; i.e. it deletes each p_i without adding any fluent in B_i, for $i = 1, \ldots, n_b$. Second, the offsets $\delta(a, s, C)$ for the heuristic $h(G|s, C)$ must be defined as:

$$\delta(a,s,C) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } a \text{ violates no \textit{disjunctive} commitment in } C \\ \max_b \max_{i=1,n_b} \min_{q\in B_i} h(q|s,C) & \text{otherwise} \end{cases} \tag{4}$$

where $\langle b, p_i, B_i\rangle$, $i = 1, \ldots, n_b$, $n_b \geq 1$, constitute the disjunctive commitments violated by action a. Finally, the commitments C' in the node $n' = \langle s', C'\rangle$ that follow the action a in node $n = \langle s, C\rangle$ are formed from C by removing the disjunctive commitments consumed by a (the set of commitments $\langle b, p_i, B_i\rangle$ with a common action b such that a adds a fluent in some B_i), by adding the disjunctive commitments made by a (as already defined), and last, by updating the rest of the disjunctive commitments in C. A disjunctive commitment $\langle b, p_i, B_i\rangle$ in C, $i = 1, \ldots, n_b$, is updated by removing from C the individual commitments $\langle b, p_i, B_i\rangle$ violated by a. Notice that at least one such commitment must remain in C if a is a helpful action according to the heuristic $h(G|s, C)$.

Landmark Graph

The overall picture for landmarks and their ordering is not too different from LAMA except that we don't deal with disjunctive landmarks, nor with a landmark heuristic. A minor difference is that we define and compute landmarks using a formulation that is a slight variation of the set-additive heuristic (Keyder and Geffner 2008; Keyder, Richter, and Helmert 2010).

The landmarks are computed as a preprocessing step from the equations below, where $L(p)$ and $L(a)$ stand for the landmarks needed in order to achieve p or apply a from the given initial state s, and $O(p)$ stands for the actions that add p:

$$L(p) = \begin{cases} \{p\} & \text{if } p \in s \\ \cap_{a\in O(p)} L(a) & \text{otherwise} \end{cases} \tag{5}$$

where

$$L(a) = \cup_{q\in Pre(a)} L(q)\ .$$

Provided that all labels $L(p)$, except for $p \in s$, are initialized to $L(p) = \perp$ ('undefined'), and that no 'undefined' label is propagated, the computation converges to labels $L(p)$ that are sound and complete relative to the delete-relaxation.

The landmarks of the problem are then those in $L(G_d)$, where G_d is the dummy goal. These landmarks are ordered by means of a directed acyclic graph such that an edge $p \rightarrow q$ means that p is a landmark for q, i.e. $p \in L(q)$, without being a landmark for another r, $r \in L(q)$.

Greedy necessary orderings (Hoffmann, Porteous, and Sebastia 2004): an edge $p \rightarrow_{gn} q$ denoting that p is greedy necessary for q (i.e. that p must be true right before q), is added if $p \in L(q)$, and all the first achievers of q have p in their preconditions. The first achievers of q are those actions a for which $q \in add(a)$ and $q \notin L(a)$.

The landmark graph is extended by adding extra edges between top goals in G, taking advantage that they must all be true at the same time. For all pairs $p, q \in G$, an edge $p \rightarrow q$ is added when all the actions adding p e-delete q[4]. This is simply because one can show then that the last action in a plan that achieves p and q jointly, must be the action that adds q.

The *set of achieved landmarks* contains initially the landmarks that are true in the initial state. Then, a landmark is added to the set when an action adds it, and is deleted from the set when an action deletes it while being greedy necessary landmark for an unachieved landmark.

The *unachieved landmarks* in a state s are the landmarks in $L(G_d)$ for the dummy goal G_d that are not in the set of achieved landmarks.

The *first unachieved landmarks* are the unachieved landmarks that are not strictly preceded by any other unachieved landmark, i.e the roots of the unachieved landmark graph.

Consistency

When a subgoal must be selected in a node n, it is chosen as the nearest first unachieved landmark that is *consistent* relative to n. The notion of consistency is adapted from the planner C3 (Lipovetzky and Geffner 2009).

A first unachieved landmark g is *consistent* in $n = \langle s, C\rangle$ if it heads a *consistent greedy chain of unachieved landmarks*. A greedy chain is a sequence of unachieved landmarks $p_1, p_2, ..., p_k$, $k \geq 1$, where p_1 is a first unachieved landmark, p_i is greedy necessary for p_{i+1}, and p_k does not precede an unachieved landmark, or precedes an unachieved landmark r, i.e., $p \rightarrow r$, but p_k is not greedy necessary for it.[5]

Intuitively, a greedy chain $p_1, \ldots, p_k$ is *consistent* when it doesn't need to be broken; i.e, when the landmark p_{i+1} can be achieved from the state s_i that results from achieving the precedent landmark p_i, *while keeping p_i true until p_{i+1}*

[4]An action e-deletes a fluent when the fluent must be false after the action, or more precisely, when the action either deletes the fluent, has a precondition that is mutex, or adds a mutex.

[5]A greedy chain can contain a single atom p_1 if p_1 complies with the conditions on p_k.

is true, $i = 1, \ldots, k-1$. Indeed, it does not make sense to choose p_1 as the next subgoal, in order to achieve then p_2, …, p_k, if this chain of causal commitments cannot be sustained.

For example, in Blocks, when $on(1,2)$ and $on(2,3)$ must be achieved starting with both blocks on the table, it doesn't make sense to adopt the 'first unachieved landmark' $hold(1)$ as a subgoal in order to achieve $on(1,2)$, and then the dummy goal G_d, as indeed, after achieving $hold(1)$, either $hold(1)$ or $on(1,2)$ will have to be *undone* in order to achieve G_d. Thus, while a greedy chain headed by a landmark p_1 provides a potential reason for selecting p_1 as the next subgoal, the notion of consistency is aimed at detecting that some of these reasons are spurious.

The definition of the conditions under which a greedy chain is consistent borrows a number of ideas from (Lipovetzky and Geffner 2009), in particular, the notion of *projected states* that provide a fast approximation of the state that results from the achievement of a given goal.

Given a chain $p_1, \ldots, p_k$, $k \geq 1$ relative to a node $n = \langle s, C\rangle$, the projected node $n_1 = \langle s_1, C_1\rangle$ is obtained from the relaxed plan π for the goal $G_1 = \{p_1\}$ from n. The state s_1 is defined as s extended with the atoms p *added* by the actions in π. Yet since some of these atoms are mutex with p_1, the process is iterated by extending the goal G_1 and the relaxed plan π, until π includes actions that delete the atoms in s_1 that are mutex with p_1; a process that can potentially add new atoms into s_1. Likewise, the set of commitments C_1 true in the projected node n_1 are those in C, but with the commitments consumed by actions in π removed.

The projected node $n_{i+1} = \langle s_{i+1}, C_{i+1}\rangle$ for the greedy chain $p_1, \ldots, p_k$ is defined in a slightly different way for $i > 1$, as while the choice of the chain makes p_1 the first unachieved subgoal, it does *not* necessarily make p_2 the second. Instead, after achieving p_1, the probe may select to achieve other landmarks and only then come back to p_2. For this reason, s_{i+1} is defined as the set of atoms reachable from s_i that are not mutex with p_{i+1}. Three type of actions a must be excluded in this reachability analysis: those with infinite offsets $\delta(a, s_i, C_i)$, those that make p_i false without making p_{i+1} true, and those with p_{i+1} in the precondition. Similarly, C_{i+1} is obtained from C_i by removing the commitments consumed by the remaining reachable actions.

Given the projected nodes $n_i = \langle s_i, C_i\rangle$ along a greedy chain $p_1, \ldots, p_k$, $i = 1, \ldots, k$, with $n_0 = \langle s, C\rangle$, the chain is *consistent* if neither $h(G_d|s_k, C_k)$ nor $h(p_i|s_{i-1}, C_{i-1})$ is infinite, for $i = 1, \ldots, k$.

Summary

Wrapping up, PROBE is a greedy best-first planner that each time that a state s is expanded, throws a probe from the node $n = \langle s, C_0\rangle$ where C_0 is the empty set of commitments. The best-first search makes the planning algorithm complete, while the probes are designed to reach the goal greedily and fast. A probe is a sequence of actions that is computed without search by selecting at each node $n = \langle s, C\rangle$ the action that is helpful to the subgoal g associated with n or the commitments C in n. A node $n = \langle s, C\rangle$ inherits the subgoal g from its parent node in the probe, except when s achieves g or n is the first node of the probe. In these two cases, the subgoal g is selected as the nearest first unachieved landmark that heads a consistent greedy chain. Probes terminate in the goal or in failure, and they are not allowed to visit states in memory (open or closed). All the states expanded by failed probes are added nonetheless to the open list of the best-first search algorithm.

Experimental Results

We compare PROBE with FF and LAMA over a broad range of IPC domains.[6] PROBE 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 it finds. 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 30 minutes or 2 GB. All action costs are assumed to be 1 so that plan cost is plan length.

Table 1 compares PROBE with FF and LAMA over 980 instances from previous IPCs. In terms of coverage, PROBE solves 21 more problems than LAMA and 73 more than FF. More remarkably, 70% of them are solved with just *one probe* (56 problems more than FF in EHC). There are several domains, like Mystery and Storage[7], where PROBE solves more problems than LAMA and FF; the largest difference to LAMA, however, being in Mprime that LAMA doesn't process well.[8] On the other hand, the largest gain of LAMA and FF over PROBE is in Sokoban, where LAMA and FF solve 12 and 13 more instances respectively.

Column #P shows the average number of probes required in each domain, which corresponds to the number of nodes expanded in the greedy best first search (not the total number of nodes shown that includes the probes). Interestingly, this number is one in most domains, and large in three domains only, Sokoban, Trucks, and Pegsol, where probes do not pay off.

A measure of the search effort is given by the number of nodes that each planner expands over the instances solved by all three planners. LAMA expands around 7 times more nodes than PROBE and FF 36 times more. In some domains this difference is larger. In Depots, for example, LAMA solves less instances than PROBE and it expands 414 times more nodes. This, however, does not mean that PROBE is faster. One reason is the use of deferred evaluation by LAMA, which leads to faster node expansions and fewer heuristic evaluations. Another one is the overhead in PROBE. Interestingly, FF is fastest in 18 out of the 30 domains, while LAMA and Probe are each fastest in 6. The

[6] FF is FF2.3, while LAMA is the version used in the 2008 IPC, except for the two Pipesworld domains, where the most recent 2010 version that fixes some problems in the parsing and preprocessing was used.

[7] A problem proved to be unsolvable by a planner, is counted as solved.

[8] LAMA has problems processing the original IPC version of Mprime where it can solve 4 instances only. There is an amended version of this domain, however, that allows LAMA to solve 35 instances, actually one more than FF and PROBE.

		FF					LAMA				PROBE					
Domain	I	S	EHC	EX	T	Q	S	EX	T	Q	S	1P	EX	#P	T	Q
Blocks World	50	42	42	9,193	0.22	**39** (9)	**50**	1,077	0.69	86 (1)	**50**	50	**40**	1.0	**0.21**	40 (2)
Cyber	30	4	4	228	**0.74**	30 (0)	**25**	73	48.48	30 (0)	24	13	**30**	111.5	1.46	30 (0)
Depots	22	**22**	19	71,649	38.28	47 (0)	20	44,738	46.58	52 (1)	**22**	14	**108**	11.8	**3.01**	**42** (6)
Driver	20	16	6	11,476	11.73	**34** (2)	**20**	2,445	1.32	37 (4)	**20**	15	**54**	2.1	**0.99**	50 (1)
Elevator	30	30	30	1,429	**1.34**	**86** (4)	30	666	3.28	88 (0)	30	25	**114**	1.2	30.81	110 (0)
Ferry	50	50	50	50	**0.01**	**28** (0)	50	108	0.18	29 (0)	50	50	**29**	1.0	0.02	29 (1)
Freecell	20	**20**	14	1,506	**2.81**	**55** (7)	**20**	2,071	19.78	64 (0)	18	7	**261**	35.1	45.45	67 (1)
Grid	5	5	5	301	**0.30**	61 (1)	5	174	6.43	**56** (1)	5	5	**59**	1.0	7.74	59 (1)
Gripper	50	50	50	102	**0.01**	**76** (0)	50	**79**	0.26	**76** (0)	50	50	101	1.0	0.06	101 (0)
Logistics	28	28	28	94	**0.01**	**41** (1)	28	97	0.25	42 (0)	28	28	**55**	1.0	0.13	55 (0)
Miconic	50	50	50	52	**0.01**	**30** (0)	50	**37**	0.15	**30** (0)	50	50	45	1.0	0.02	45 (0)
Mprime	35	**34**	34	23	**0.03**	**6** (1)	4	12	3.72	**6** (0)	**34**	33	**7**	1.0	2.62	7 (0)
Mystery	30	18	15	258	**0.08**	7 (0)	22	238	2.36	**6** (4)	**25**	23	**8**	1.1	1.21	8 (0)
Openstacks	30	30	30	504	**0.46**	**136** (0)	30	124	3.03	145 (0)	30	30	**121**	1.0	20.22	139 (0)
Openstacks-IPC6	30	30	30	968	**0.59**	**156** (0)	30	146	3.68	159 (0)	30	30	**139**	1.0	54.76	158 (0)
Parc-Printer	30	**30**	21	173	**0.03**	32 (0)	24	409	0.41	34 (0)	27	21	**49**	9.7	0.26	**31** (0)
Pegsol	30	**30**	0	15,287	1.35	**34** (0)	**30**	5,174	**1.34**	35 (0)	29	1	**1,681**	864.7	2.10	**34** (0)
Pipesworld-No-Tan	50	35	17	3,540	0.45	**28** (5)	44	1,363	1.04	37 (1)	**45**	19	**65**	6.4	**0.35**	33 (5)
Pipesworld-Tan	50	22	4	46,189	62.23	**30** (8)	39	40,015	**32.41**	31 (2)	**41**	16	**1,055**	108.7	59.14	55 (5)
PSR-Small	50	41	0	39,533	60.96	**17** (1)	**50**	276	0.89	**17** (0)	**50**	0	**70**	30.8	**0.07**	20 (0)
Rovers	40	40	40	10,341	26.97	**100** (4)	40	1,750	**13.44**	106 (1)	40	38	**114**	1.1	28.16	113 (0)
Satellite	20	20	20	389	**0.10**	**38** (0)	20	412	0.90	39 (1)	20	20	**41**	1.0	0.86	41 (0)
Scanalyzer	30	**30**	22	1,905	**1.89**	24 (1)	28	257	8.52	24 (2)	28	26	**39**	2.8	6.15	24 (4)
Sokoban	30	**27**	0	19,355	**0.82**	141 (2)	26	16,066	3.52	**138** (4)	14	0	**12,027**	11,120.6	96.71	160 (0)
Storage	30	18	3	261,299	49.90	16 (0)	18	3,645	1.62	20 (0)	**21**	15	**15**	2.5	**0.08**	**15** (6)
TPP	30	28	28	28,388	42.41	122 (0)	**30**	1,340	**6.91**	**104** (8)	**30**	30	**119**	1.0	20.88	119 (0)
Transport	30	29	29	45,593	133.52	28 (1)	**30**	4,964	**41.23**	27 (0)	**30**	24	**157**	1.2	42.27	**26** (4)
Trucks	30	11	6	135,863	5.66	**23** (0)	**16**	**169**	**0.61**	24 (0)	9	0	2,762	2,818.4	20.55	26 (0)
Woodworking	30	17	12	1,329	**0.26**	117 (1)	**30**	7,040	5.84	**100** (15)	**30**	30	**31**	1.0	5.45	154 (0)
Zeno Travel	20	20	18	148	**0.13**	**31** (13)	20	482	3.55	36 (1)	20	20	**50**	1.0	6.21	50 (0)
Total	980	827	627	23,572	14.77	54	879	4,515	8.75	56	900	683	648		15.26	61
Percentage		84%	64%				89%				92%	70%				

Table 1: PROBE vs. FF and LAMA on instances of previous IPCs: *I* is number of instances, *S* is number of solved instances, *EHC* is number of instances solved by EHC, *EX* is avg. number of expanded nodes, *1P* is number of instances solved with *one* probe, *#P* is avg. number of probes triggered, *T* is avg. time in sec. and *Q* is avg. plan length. *EX*, *T* and *Q* are reported for problems solved by all three planners. In parenthesis is the number of problems where each planner produces solutions that are at least 10% better than the other two planners.

average plan length of the instances solved by the three planners is 61 for PROBE, 56 for LAMA and 54 for FF. PROBE is worst in quality in *Sokoban* and *Gripper*, while best in *Depots* and *Blocks*.

We have also evaluated the impact of the different design decisions made in PROBE. The results are summarized in Table 2 where the columns show the percentage of problems solved, the percentage of problems solved with a single probe and the avg. plan length and time that results from dropping some *feature* from PROBE; namely, the *probes* themselves, the subgoal *consistent tests*, the *subgoaling mechanism* itself, and the *commitments*.[9] In this table, the averages are computed over the problems solved by PROBE with all these features, and thus they differ from the averages in the previous table computed over the problems solved by the three planners. As it can be seen from the table, dropping the probes from PROBE, i.e., making it a standard greedy BFS planner, reduces the coverage from 92% to 75%, while the times increase by a factor of 3. The removal of consistent tests and the removal of the whole subgoaling mechanism, in turn, do not affect coverage as much, but reduce the percentage of problems solved with a single probe from 67% to 40% and 44%, while increasing times and lengths by roughly 50% and 25% respectively. Likewise, if only commitments are dropped, the loss is mainly on the avg. length of plans that jumps up 26%.

From these figures, a number of conclusions can be drawn. First, the use of probes helps significantly along all relevant dimensions. Second, *subgoaling helps as well but only when used in combination with the consistency tests* (degradation from turning off consistency is similar to degradation from turning off the whole subgoaling mechanism). Third, commitments help but mainly to improve the quality of the resulting plans; something that is achieved by keeping track of the reasons for the introduction of the actions in the plan.

[9]The removal of the subgoaling mechanism means that the heuristic minimization used to select the action to do next is not done for the selected subgoal, but over all possible first subgoals.

Feature Off	S	1P	Q	T
None	**92%**	**70%**	**67.0**	**34.8**
Probes	75%	–	71.0	99.6
Consistency	91%	40%	91.4	56.9
Subgoaling	86%	44%	80.7	55.2
Commitments	90%	63%	85.0	39.0

Table 2: Ablation Study. The columns indicate the feature of PROBE that was turned off, the % of problems solved (S) and solved by a single probe (1P), and the avg. plan length (Q) and time (T) in seconds. The averages are computed over all problems solved by PROBE.

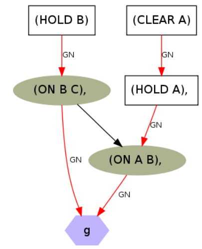

Figure 1: The landmark graph for Sussman's anomaly. Top goals and their landmarks are shown as elliptical and rectangular nodes respectively, and the arrows represent ordering relations between them. Greedy necessary orderings marked 'GN'

Example

PROBE is a 'fine-grained' planner that can solve many problems without search, and thus it is illustrative to see its behavior over concrete instances.

The Sussman Anomaly is a Block World problem that starts with blocks b and a on the table, and c on top of a, and requires some form of goal interleaving for achieving the goals b on c and a on b. Indeed no goal can be tackled first while leaving the other goal aside; progress towards the two subgoals needs to be interleaved, which can defeat naive serialization schemes.

The landmark graph generated for this problem is shown in Figure 1. The goal *on(b,c)* must be achieved before *on(a,b)*, as the actions that add the first goal e-delete the second. The landmarks of *on(a,b)* are $hold(a)$, which is greedy necessary for *on(a,b)*, and $clear(a)$ which is greedy necessary for $hold(a)$. The goal *on(b,c)* is preceded only by the greedy necessary landmark $hold(b)$. The two goals are in turn greedy necessary for the dummy end goal g.

As described above, the first probe is launched from the initial state. First, it must select a subgoal. The selection process computes the set of consistent first unachieved landmarks and chooses the one with the lowest heuristic value. In this case, the only consistent landmark is $clear(a)$. The other first unachieved landmark $hold(b)$ is not consistent, as the heuristic over the state s_2 that results from the projection when $on(b,c)$ is achieved in the greedy chain $hold(b), on(b,c), g$, is infinite, meaning that from that state $on(a,b)$ can't be achieved by maintaining $on(b,c)$.

Once the subgoal $clear(a)$ is selected, the action selection process is triggered. There is one helpful action with respect to $hold(a)$, $unstack(c,a)$, which leaves the subgoal at distance 0. The action $a_0 = unstack(c,a)$ adds the commitment

$$\langle a_0, clear(a), \{hold(a)\}\rangle$$

that can only be consumed by the action $pickup(a)$ given that a is on the table. Notice that committing to maintain $clear(a)$ until $hold(a)$ is achieved results in all possible $stack(X,a)$ actions being penalized with an offset by the heuristic.

In the resulting node, goal selection is triggered because the previous subgoal has been made true in the parent node. Among the two first unachieved landmarks $hold(a)$ and $hold(b)$, only the latter is consistent. $hold(a)$ is not consistent as $h(on(b,c)|s_3, C_3) = \infty$, where the pair s_3, C_3 represents the state and set of commitments resulting from the projection along the chain $p_1, p_2, p_3 = hold(a), on(a,b), g$. Once $hold(b)$ is selected as the new subgoal, the helpful actions with respect to $hold(b)$ and $hold(a)$ are computed. Notice that though $hold(a)$ is not the current subgoal, helpful actions are computed for it as well, as it is a goal of one of the active commitments. The only action that respects the current commitments is then $a_1 = putdown(c)$, adding the commitment

$$\langle a_1, freearm, \{hold(a), hold(b)\}\rangle \ .$$

As the current subgoal is not yet achieved in the resulting node, goal selection is skipped and the action selection procedure computes the helpful actions with respect to $hold(b)$ and $hold(a)$. There are two actions: $pickup(b)$ which leaves the subgoal at distance 0, and $pickup(a)$ that leaves the subgoal at distance 2. Therefore, $a_2 = pickup(b)$ is selected, consuming the last commitment and adding instead the commitment

$$\langle a_2, hold(b), \{on(b,c)\}\rangle \ .$$

In the resulting node, goal selection is triggered again, selecting the top goal $on(b,c)$ and discarding $hold(a)$, because it still does not begin a consistent greedy chain. The only helpful action for $on(b,c)$ and $hold(a)$ is $a_3 = stack(b,c)$, which consumes the last commitment, and adds the disjunctive commitment

$$\langle a_3, on(b,c), \{g\}\rangle \vee \langle a_3, freearm, \{hold(a)\}\rangle \ .$$

The probe continues, selecting the only possible new subgoal $hold(a)$, which is consistent because $on(b,c)$ is already true in the current state. It then selects the helpful action $a_4 = pickup(a)$ that consumes the two existing commitments (a_0, a_3), and adds

$$\langle a_4, hold(a), \{on(a,b)\}\rangle \ .$$

Finally the subgoal $on(a,b)$ is selected, and the helpful action $a_5 = stack(a,b)$ is applied, consuming the last commitment and adding $\langle a_5, on(a,b), \{g\}\rangle$. The probe ends successfully with the selection of the End action that adds that last landmark, that stands for the dummy goal g.

Conclusions

We have formulated and tested a new dual search architecture for planning based on the notion of probes: single action sequences constructed greedily but carefully, that can quickly get deep into the state space, terminating in the goal or in failure. The probes are used as part of a greedy best-first search algorithm that throws a single probe from every state that is expanded. We have shown that most IPC domains are solved with a single probe, while in a few difficult domains such as Sokoban and Trucks, probes do not help and introduce overhead. Overall, the performance of the planner is comparable with state-of-the-art planners such as FF and LAMA.

The design of probes uses and extends a number of techniques developed in modern planners that go well beyond the use of heuristic functions to guide the search. They include helpful actions, landmarks, causal commitments, subgoals, and consistency tests, all of which help in the greedy selection of the subgoal to achieve next, and the actions needed to reach it.

From the success of probes and their computation, in which problems are mapped into a *series of subgoals* that are heuristically computed along with the probes, two conclusions can be drawn. The first is that most of the classical benchmarks admit good serializations of the landmarks under which the solution of the problems becomes simple. The second is that while not every serialization is good, the mechanisms in PROBE and in particular the consistency tests, appear to find good ones.[10] These observations raise two questions that we would like to address in the future. The first is which methods are good for finding good serializations when they exist. PROBE implements one such method but it's not necessarily the best such method. The second question is which methods are good for finding and exploiting serializations in problems that have good but no perfect decompositions. The 8-puzzle is an example of this situation: one can place the tile 1 in position 1, the tile 2 in position 2, but then one needs to undo this last subgoal, in order to have tiles 2 and 3 at their target positions.

The ideas of goal serialization and problem decomposition have received a lot of attention in search and in the early days of planning (Korf 1987), and it may be worth revisiting those ideas equipped with the techniques that have been developed more recently in planning research. The challenge is to explicitly recognize and exploit the structure of problems that are nearly-decomposable, even if they are not perfectly decomposable. Indeed, planning used to be defined originally as being mainly concerned with those problems (Newell and Simon 1963). While the notion has practically disappeared from the modern language of planning, it is still very much there: classical planners do best on those problems[11], simply because the heuristics used, like delete-relaxation heuristics, assume that problems are in fact decomposable. Nonetheless, there is the possibility that modern planners could do better still on nearly-decomposable problems, if they would recognize and exploit the good but not perfect serializations that such problems hide.

[10] Landmarks are used in a subgoaling scheme in (Hoffmann, Porteous, and Sebastia 2004), but the results do not appear to be as good. One possible explanation for this, is that no additional inference is made to distinguish good serializations from bad ones.

[11] Planners can solve other problems too, but expanding much larger number of nodes, and not scaling up that well.

Acknowledgements

We thank the anonymous reviewers for useful comments. This work is partially supported by grants TIN2009-10232, MICINN, Spain, and EC-7PM-SpaceBook.

References

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

Harvey, W., and Ginsberg, M. 1995. Limited discrepancy search. In *Proc. IJCAI-95*.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *JAIR* 22:215–278.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *JAIR* 20:291–341.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In *Proc. ECAI-08*.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In *Proc. ECAI-10*.

Korf, R. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33(1):65–88.

Langley, P. 1992. Systematic and non-systematic search strategies. In *Proc. AIPS-92*.

Lipovetzky, N., and Geffner, H. 2009. Inference and decomposition in planning using causal consistent chains. In *Proc. ICAPS-09*.

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

McDermott, D. 1996. A heuristic estimator for means-ends analysis in planning. In *Proc. AIPS-96*.

Nakhost, H., and Müller, M. 2009. Monte-Carlo exploration for deterministic planning. In *Proc. IJCAI-09*.

Newell, A., and Simon, H. 1963. GPS: a program that simulates human thought. In Feigenbaum, E., and Feldman, J., eds., *Computers and Thought*. McGraw Hill. 279–293.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:122–177.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proc. AAAI-08*.

Tate, A. 1977. Generating project networks. In *Proc. IJCAI-77*.

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

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proc. ICAPS-04*.