Local Search Topology in Planning Benchmarks: An Empirical Analysis

Jörg Hoffmann
Institute for Computer Science
Albert Ludwigs University
Georges-Köhler-Allee, Geb. 52
79110 Freiburg, Germany

Abstract

Many state-of-the-art heuristic planners derive their heuristic function by relaxing the planning task at hand, where the relaxation is to assume that all delete lists are empty. Looking at a collection of planning benchmarks, we measure topological properties of state spaces with respect to that relaxation. The results suggest that, given the heuristic based on the relaxation, many planning benchmarks are simple in structure. This sheds light on the recent success of heuristic planners employing local search.

1 Introduction

In the last two years, planning systems based on the idea of heuristic search have been very successful. At the AIPS-1998 planning systems competition, HSP1 compared well with the other systems [McDermott, 2000], and at the AIPS-2000 competition, out of five awarded fully automatic planners, FF and HSP2 were based on heuristic search, while another two, Mips and STAN, were hybrids that incorporated, amongst other things, heuristic search [Bacchus and Nau, 2001].

Interestingly, four of these five planners use the same base approach for deriving their heuristic functions: they relax the planning task description by ignoring all delete lists, and estimate, to each search state, the length of an optimal relaxed solution to that state. This general idea has first been proposed by Bonet et al. [1997]. The length of an optimal relaxed solution would yield an admissible heuristic. However, as was proven by Bylander [1994], computing the optimal relaxed solution length is still NP-hard. Therefore, Bonet et al. introduced a technique for approximating optimal relaxed solution length, which they use in both versions of HSP [Bonet and Geffner, 2001]. The heuristic engines in FF [Hoffmann, 2000] and Mips [Edelkamp, 2000] use different approximation techniques.

Three of the above planners, HSP1, FF, and Mips, use their heuristic estimates in variations of local search algorithms, where the search space to a task is the state space, i.e., the space of all states that are reachable from the initial state. Now, the behavior of local search depends crucially on the problem structure, i.e., on the topology of the search space. Thus, the success of these heuristic planners on many planning tasks gives rise to the suspicion that those task's state spaces have a simple structure with respect to relaxed goal distances. In this paper, we shed light on that suspicion. Following Frank et al. [1997], we define a number of structural phenomena in search spaces under heuristic evaluation, impacting the performance of local search algorithms. We compute the optimal relaxed solution length to reachable states in small planning tasks, and measure structural properties. Our results suggest that, in fact, the tasks contained in many benchmark planning domains have a simple state space topology, at least when using the optimal relaxed heuristic. To give an example of how this observation carries over to the approximation techniques used by existing heuristic planners, we apply the same technique of data collection to the FF heuristic. As it turns out, the results are similar. Specifically, it follows that FF's search algorithm is a polynomial solving mechanism in a number of planning benchmark domains, under the hypothesis that the larger instances behave similar to the smaller ones.

Section 2 introduces our general approach, Section 3 gives the basic definitions. Sections 4 and 5 define structural phenomena in search spaces under heuristic evaluation, and give empirical data. Section 6 summarizes the results in a taxonomy for planning domains. Section 7 applies the methodology to the FF heuristic. Section 8 concludes and gives an outlook on further research.

2 General Approach

In our experiments, we used solvable planning tasks only, as we are interested in finding out why local search can succeed so quickly on many benchmark tasks. We looked at instances from 20 different STRIPS and ADL benchmark domains. Due to space restrictions, we only present the results for the domains used in the competitions here, as those domains are well known in the planning community.

To obtain data on how planning tasks behave with respect to the relaxation, rather than with respect to any of the approximation techniques used by existing heuristic planners, we consider the optimal relaxed solution length as our heuristic. As determining that optimal length is NP-hard, it can only be computed for small planning instances. We build an explicit state space representation to such instances, and look at the topology in detail. This yields a clear picture of the fundamental structural differences between instances from different planning domains. In that context, we state some hypotheses. A piece of future work is to verify those.

In total, the competitions featured 13 STRIPS and ADL domains: *Assembly*, *Blocksworld*, *Freecell*, *Grid*, *Gripper*, *Logistics*, *Miconic-ADL*, *Miconic-SIMPLE*, *Miconic-STRIPS*, *Movie*, *Mprime*, *Mystery*, and *Schedule*. In 11 of these domains, we used random task generation software to produce small instances, at least 100 per domain. In *Gripper*, there is only one instance of each size: n balls to be transported. In *Movie*, every instance of the AIPS-1998 suite was small enough to be looked at in detail.

Sometimes, we depict scaling behavior. As our instances are all quite small anyway, we need, for that purpose, a finer distinction between instances than obvious criteria like the number of objects. We define the difficulty of a task to be the length of an optimal solution plan, and order our instances within any domain by increasing difficulty. Except in the *Movie* domain (where all instances in the AIPS-1998 suite have the same difficulty), larger instances are on average more difficult than smaller ones. The maximal difficulty of any instance we could look at is 21 in the *Gripper* domain, 20 in the *Assembly* and *Logistics* domains, 18 in *Grid*, and 16 in the *Blocksworld*. In *Movie*, all instances have difficulty 7, and in the remaining domains our maximal difficulty ranges from 10 to 14.

3 Basic Definitions

The competition domains contain tasks specified in the STRIPS and ADL languages. In both cases, a planning task $\mathcal{P}$ is specified in terms of a set of objects O, an initial state $\mathcal{I}$, a goal formula $\mathcal{G}$, and a set of operator schemata $\mathcal{O}$. $\mathcal{I}$, $\mathcal{G}$, and $\mathcal{O}$ are based on a collection of predicate symbols. Planning tasks from the same domain share the same sets of predicate symbols and operator schemata. Instantiating the operator schemata with all objects yields the actions A to the task. States are sets of logical atoms, i.e., instantiated predicates. Any action has a precondition, which is a formula that must hold in a state for the action to be applicable. Also, an action has an add- and a delete-list. These are sets of atoms, where each atom has a condition formula attached to it (in STRIPS, these condition formulae are trivially TRUE). If an action is applied, the atoms with satisfied condition in the add list are added to the state, and those with satisfied condition in the delete list are removed from the state. A plan is a sequence of actions that, when successively applied to the initial state, yields a state that satisfies the goal formula.

Ignoring the delete lists simplifies a task only if all formulae are negation free. In STRIPS, this is the case by definition. In general, for a fixed domain, any task can be polynomially transformed to have that property: compute the negation normal form to all formulae (negations only in front of atoms), then introduce for each negated atom $\neg B$ a new atom not-B and make sure it is TRUE in a state iff B is FALSE [Gazen and Knoblock, 1997]. In the following, we assume formulae to be negation free. We will investigate properties of the optimal relaxed heuristic h^+. For any state s in a planning task with actions A and goal condition $\mathcal{G}$, the *relaxed task to* s is the task defined by the same goal condition $\mathcal{G}$, the initial state s, and the action set A', which is identical to A except that all delete lists are empty. Then, $h^+(s)$ is the length of a shortest plan that solves the relaxed task to s, or $h^+(s) = \infty$ if there is no such plan.

We will be looking at the topology of search spaces with heuristic evaluation. The structural properties we will introduce do not depend on the planning framework. We therefore define them in a general manner, embedding planning state spaces as a special case.

Definition 1 *A* search space *is a* 4*-tuple* (S, E, G, s_0)*, where* S *is the set of* states*,* $E \subseteq S \times S$ *are the* state transitions*,* $\emptyset \neq G \subseteq S$ *are the* goal states*, and* $s_0 \in S$ *is the* initial state.

Given a planning task, the search space we look at is what is usually referred to as the *state space*. There, s_0 is simply the initial state $\mathcal{I}$ of the task. S is the set of states that are reachable from the initial state by successively applying actions from A, and E contains all pairs (s, s') where one action, executed in s, yields the state s'. G is the set of all states that satisfy the goal condition. Looking only at solvable instances, there is at least one such state.

Definition 2 *Given a search space* (S, E, G, s_0)*. The* goal distance *of* $s \in S$ *is*

$$gd(s) := min\{dist(s, s') \mid s' \in G\}$$

The distance $dist(s, s')$ between any two states is the length of a shortest path from s to s' in the directed graph given by S and E, or $dist(s, s') = \infty$ if there is no such path. Heuristic functions approximate gd.

Definition 3 *Given a search space* (S, E, G, s_0)*. A* heuristic *is a function* $h : S \mapsto N_0 \cup \{\infty\}$*, such that* $h(s) = 0 \Leftrightarrow gd(s) = 0$.

We require that a heuristic recognizes goal states, yielding $h(s) = 0$ if and only if $gd(s) = 0$, which is equivalent to $s \in G$. We allow heuristics to return $h(s) = \infty$, as search spaces can contain *dead ends*.

4 Dead Ends

Because state transitions in a search space are, in general, directed, there can be states from which no goal state is reachable.

Definition 4 *Given a search space* (S, E, G, s_0)*. A state* $s \in S$ *is a* dead end*, if* $gd(s) = \infty$.

If a local search algorithm runs into a dead end, it is lost. A heuristic function can return $h(s) = \infty$ to indicate that s might be a dead end. Desirably, it does so only on states s that really are dead ends.

Definition 5 *Given a search space* (S, E, G, s_0) *with a heuristic* h*.* h *is* completeness preserving*, if* $h(s) = \infty \Rightarrow gd(s) = \infty$.

With a completeness-preserving heuristic, we can safely prune states where $h(s) = \infty$. For planning tasks, if a task can not be solved even when ignoring the delete lists, then the task is unsolvable. Therefore, the h^+ function is completeness preserving. For the rest of the paper, we only consider those states where the heuristic value is less than ∞.

Definition 6 *Given a search space* (S, E, G, s_0) *with a completeness-preserving heuristic* h*. The* relevant part *of the search space is* $\{s \in S \mid h(s) < \infty\}$.

Any search space with heuristic evaluation falls into one of the following four classes, with respect to dead ends.

Definition 7 *Given a search space* (S, E, G, s_0) *with a completeness-preserving heuristic* h*. The search space is*

1. undirected*, if* $\forall (s, s') \in E : (s', s) \in E$
2. harmless*, if* $\exists (s, s') \in E : (s', s) \notin E$*, and* $\forall s \in S : gd(s) < \infty$
3. recognized*, if* $\exists s \in S : gd(s) = \infty$*, and* $\forall s \in S : gd(s) = \infty \Rightarrow h(s) = \infty$
4. unrecognized*, if* $\exists s \in S : gd(s) = \infty \wedge h(s) < \infty$

For each of our planning instances, we verified which of the above classes the state space belonged to. We say that a domain belongs to class i if the state spaces of all our instances belong to a class $j \leq i$, and at least one instance belongs to class i. We found the following.

1. *Blocksworld*, *Gripper*, and *Logistics* have undirected graphs.
2. *Grid*, *Miconic-STRIPS*, *Miconic-SIMPLE*, and *Movie* are directed, but do not have dead ends.
3. In *Assembly* and *Schedule*, all dead ends are recognized.
4. *Freecell*, *Miconic-ADL*, *Mprime*, and *Mystery* contain unrecognized dead ends.

In addition to our empirical analysis, the results from 1. and 2. can be shown analytically. For undirected graphs, such a method is described by Koehler and Hoffmann [2000]. For the domains in class four, it is also interesting to see how many unrecognized dead ends there are. We measure the percentage of such states in the relevant part of the state space, see Figure 1.

Domain	I_0	I_1	I_2	I_3	I_4
Freecell	0.0	1.1	1.2	2.6	3.0
Miconic-ADL	0.0	0.0	2.5	9.7	9.8
Mprime	18.8	29.3	50.0	58.0	69.6
Mystery	19.5	37.9	54.0	66.4	84.6

Figure 1: Percentage of unrecognized dead ends in the relevant part of the state space. Mean values for increasing task difficulty in different domains.

For each single domain in Figure 1, the sequence of columns gives a picture of how the values develop with increasing task difficulty. In each domain, the tasks are divided into five groups. The difficulty of a task in group i lies within interval I_i, where $I_0 \dots I_4$ divide our range of difficulty in that domain into five parts of same size. Note that the intervals I_i are different for each domain, so the values within a column are not directly comparable.

As Figure 1 shows, the *Mprime* and *Mystery* tasks can contain a lot of unrecognized dead ends, and have the tendency to contain more of such dead ends the more difficult they get. For *Freecell* and *Miconic-ADL*, we can not conclude much more than that there can be unrecognized dead ends. It seems that the percentage grows with task difficulty, and that tasks with high percentage are out of the range of difficulty we could look at.

5 Search Space Topology

For SAT problems, the topology of search spaces with respect to the behavior of local search has been investigated by Frank et al. [1997]. As the basis of their work, Frank et al. formally define a partitioning of the search space into plateaus of different kinds. For our purposes, we extend their definitions to deal with our general notion of search spaces with heuristic evaluation, where edges can be directed.

Definition 8 *Given a search space* (S, E, G, s_0) *with a completeness-preserving heuristic* h*. For* $l \in N_0 \cup \{\infty\}$*, a* plateau P of level l *is a maximal subset of* S *for which the induced subgraph in* (S, E) *is strongly connected, and* $h(s) = l$ *for each* $s \in P$*.*

Plateaus are regions that are equivalent under reachability aspects, and look the same from the point of view of the heuristic function. Obviously, each state s lies on exactly one plateau.

Definition 9 *Given a search space* (S, E, G, s_0) *with a completeness-preserving heuristic* h*, and a plateau* P*. A state* $s \in P$ *is an* exit of P*, if there is a state* $s' \notin P$ *such that* $(s, s') \in E$ *and* $h(s') \leq h(s)$*.* s *is an* improving *exit, if, for at least one such* s'*,* $h(s') < h(s)$*.*

Exits are states from which one can leave a plateau without increasing the value of the heuristic function. In undirected graphs, like are considered by Frank et al. [1997], leaving a plateau implies changing the value of the heuristic function, so all exits are improving there. According to the proportion of exits on a plateau, Frank et al. divide plateaus into four classes: local minima, benches, contours, and global minima. Taking account of directed edges, we have two types of exits, and define the following six different classes.

Definition 10 *Given a search space* (S, E, G, s_0) *with a completeness-preserving heuristic* h*.*

1. *A* recognized dead end *is a plateau* P *of level* $h = \infty$*.*
2. *A* local minimum *is a plateau* P *of level* $0 < h < \infty$ *that has no exits.*
3. *A* plain *is a plateau* P *of level* $0 < h < \infty$ *that has at least one exit, but no improving ones.*
4. *A* bench *is a plateau* P *of level* $0 < h < \infty$ *that has at least one improving exit, and at least one state that is not an improving exit.*
5. *A* contour *is a plateau* P *of level* $0 < h < \infty$ *that consists entirely of improving exits.*
6. *A* global minimum *is a plateau* P *of level* 0*.*

Each plateau belongs to exactly one of the above classes. In our solvable instances, global minima are exactly the plateaus of level 0. With a completeness-preserving heuristic, recognized dead ends are irrelevant, and can be ignored. From local minima, there is no direct way of getting closer to the goal.

From benches, there is. From contours, one can get closer immediately. Plains behave as a kind of entrance to either local minima or benches.

Definition 11 *Given a search space* (S, E, G, s_0) *with a completeness-preserving heuristic* h. *A* flat path *is a path where all states on the path have the same heuristic value. For a plateau* P*, the* flat region $FR(P)$ from P *is the set of all plateaus* P' *such that there is a flat path from some* $s \in P$ *to some* $s' \in P'$.

For a plain P, if there is at least one bench or contour in $FR(P)$, then P behaves similar to a bench, with at least one improving exit being within reach. Otherwise, starting in P, without increasing the value of the heuristic function, one will inevitably end up in a local minimum.

Definition 12 *Given a search space* (S, E, G, s_0) *with a completeness-preserving heuristic* h. *A plain* P leads to benches *if* $FR(P)$ *contains some bench or contour. Otherwise,* P leads to local minima.

Based on the above definitions, and using an explicit search space representation, one can measure all kinds of structural parameters. Due to space restrictions, we only discuss some of the most interesting parameters here.

5.1 Local Minima

First, we are interested in the percentage of states that lie on local minima. Before doing this, we need to take a closer look at the definition of local minima. These are flat regions where all neighbors have higher evaluation. Stepping on to one of these neighbors does not necessarily improve the situation, though: it might be, for example, that the only exits on that neighbor lead back to the local minimum. In general, a local minimum is only the bottom of a valley, where what we really want to know about is the whole valley. Valleys are characterized by the property that one can not reach a goal state without increasing the value of the heuristic function.

Definition 13 *Given a search space* (S, E, G, s_0) *with a completeness-preserving heuristic* h. *A state* $s \in S$ has a full exit path*, if there is a path from* s *to a goal state* s' *such that the heuristic value of the states on the path decreases monotonically.*

One state on a plateau has a full exit path if and only if all states on that plateau do so. If a plateau has no full exit paths, then it is part of a valley.

Definition 14 *Given a search space* (S, E, G, s_0) *with a completeness-preserving heuristic* h. *A* valley *is a maximal set* V *of plateaus such that no* $P \in V$ *has full exit paths, no* $P \in V$ *is a recognized dead end, and for all* $P, P' \in V$, P *is strongly connected to* P'.

The existence of valleys is, in any search space, equivalent to the existence of local minima. It turns out that, in 7 of the 13 competition domains, the state spaces of all our instances do not contain any local minima at all.

Hypothesis 1 *Let* $\mathcal{P}$ *be a planning task from any of the Assembly, Grid, Gripper, Logistics, Miconic-SIMPLE, Miconic-STRIPS, or Movie domains. Then, the state space of* $\mathcal{P}$ *does not contain any local minima under evaluation with* h^+.

In Figure 2, we show the mean percentage of states on valleys for those domains where we found local minima.

Domain	I_0	I_1	I_2	I_3	I_4
Blocksworld	9.8	28.1	37.5	50.1	55.6
Freecell	0.0	1.1	1.2	2.6	3.0
Miconic-ADL	0.0	0.0	2.5	9.7	9.8
Mprime	18.8	29.9	50.7	58.5	70.7
Mystery	19.5	38.5	55.1	68.0	89.4
Schedule	20.0	25.6	36.5	31.8	35.3

Figure 2: Percentage of states on valleys in the relevant part of the state space. Mean values for increasing task difficulty in different domains.

Any unrecognized dead end state lies in a valley. Therefore, the percentage of valleys is at least as high as the percentage of unrecognized dead ends for the domains shown in Figure 1. In *Freecell* and *Miconic-ADL*, the states on valleys are exactly the unrecognized dead ends in all our examples. In *Mprime* and *Mystery*, there can be more valley states. In *Blocksworld* and *Schedule*, values seem to approach an upper limit on our most difficult tasks. Computing maximum instead of the mean values shown in Figure 2, we found that some of our *Schedule* tasks contain up to 74.1% valley states. In our *Blocksworld* suite, however, the maximum valley percentage is constantly 59.3%, irrespective of difficulty.

5.2 Contours

We also measure the average percentage of states lying on contours that are not part of a valley—regions in the state space that are dominated by such contours are likely to be passed quickly by a local search algorithm. In *Movie*, the percentage is constantly 98.4%. In *Assembly*, *Logistics*, *Miconic-SIMPLE*, *Miconic-STRIPS*, and *Schedule*, between 30% and 60% of the relevant state space lie on such contours in our examples, and there is no clear tendency that the values decrease with task difficulty. In the remaining 7 domains, there is such a tendency. Values are particularly low in the *Blocksworld*, going down to 3.8% in our most difficult tasks.

5.3 Benches

For benches, the percentage of states alone is not a very informative parameter, as any plateau is a bench given it has at least one improving exit. What really matters is, how difficult is it to find such an exit? Possible criteria for this are the size of benches, or the proportion of improving exits. Here, we define another criterion that is—as will be shown in the next section—especially relevant for FF's search algorithm. The criterion is named *maximal exit distance*. We measure that distance for what we call *bench-related* plateaus. These are benches, and plains leading to benches. Recall Definitions 11 and 12.

Definition 15 *Given a search space* (S, E, G, s_0) *with a completeness-preserving heuristic* h. *For a state* s *on a bench-related plateau, the* exit distance $ed(s)$ *is the length of* *a shortest flat path from s to some s' such that there is some s'' with $(s', s'') \in E, h(s'') < h(s')$. The maximal exit distance of a bench-related plateau P is $med(P) := max\{ed(s) \mid s \in P\}$.*

The maximal exit distance in a search space is the maximum over the maximal exit distances of all bench-related plateaus, or 0 if there are no bench-related plateaus. It turns out that, in 5 of the competition domains, the maximal exit distance is constant across all our examples.

Hypothesis 2 *To any of the Gripper, Logistics, Miconic-SIMPLE, Miconic-STRIPS, or Movie domains, there is a constant c, such that, for all tasks $\mathcal{P}$ in that domain, the maximal exit distance in the state space of $\mathcal{P}$ is at most c under evaluation with h^+.*

In the listed domains, all our examples have maximal exit distance 1, so the constant $c = 1$ fulfills the hypothesized property there. The crucial point is that, in those 5 domains, there apparently is an upper limit to the maximal exit distance. In contrast to this, computing mean values, we found that the mean maximal exit distance grows with difficulty in our suites from 7 of the remaining 8 domains. In *Mprime*, mean values show a lot of variance, making it hard to draw any conclusions. See Figure 3.

Domain	I_0	I_1	I_2	I_3	I_4
Assembly	0	1.2	1.6	2.4	3.5
Blocksworld	2.8	3.3	3.8	4.9	5.0
Freecell	0	0	1	1	1.5
Grid	2.6	3.2	3.1	3.8	4.3
Miconic-ADL	1	1	1	1.4	2.3
Mprime	1.9	2.6	2.1	2.0	2.2
Mystery	1.1	1.4	1.5	1.2	2.3
Schedule	1	1	1.2	1.9	2.0

Figure 3: Maximal exit distance. Mean values for increasing task difficulty in different domains.

6 A Planning Domain Taxonomy

Our approach divides planning domains into a taxonomy of different classes with respect to the h^+ heuristic, depending on which dead end class they belong to, whether there can be local minima, and whether there is an upper limit to the maximal exit distance. See a schematic overview of our results in Figure 4.

Remember that the existence of unrecognized dead ends implies the existence of valleys, which implies the existence of local minima. The overview in Figure 4 gives an appealing impression of the kind of domains that state-of-the-art heuristic planners work well on: The “simple” domains are in the left bottom corner, while the “demanding” ones are in the top right corner. In fact, in the AIPS-2000 competition, the *Freecell* and *Miconic-ADL* domains constituted much more of a problem to the heuristic planners than, for example, the *Logistics* domain did.

The majority of the competition domains lie on the “simple” left bottom side of our taxonomy. In fact, this phenomenon gets even stronger when looking at other commonly

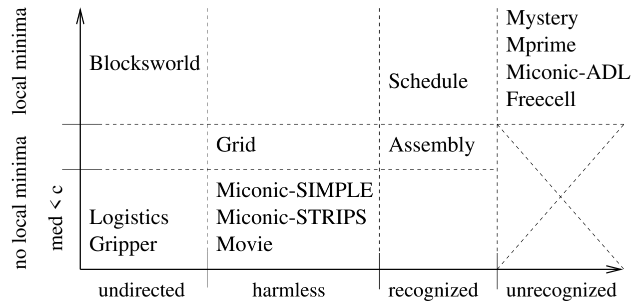

Figure 4: A taxonomy for planning domains, overviewing our results.

used planning benchmark domains: From our 20 domains, 14 do not exhibit any local minima. In 8 of those 14 domains, the mean maximal exit distance does not grow with difficulty. In the *Briefcaseworld*, for example, the distance is apparently bounded by $c = 3$.

For domains without local minima and with bounded maximal exit distance, we can be precise about simplicity. Consider the following algorithm, working on a search space (S, E, G, s_0) with a heuristic h.

$s := s_0$
while $h(s) \neq 0$ **do**
  do breadth first search for s', $h(s') < h(s)$
  $s := s'$
endwhile

This algorithm has been termed Enforced Hill-climbing by Hoffmann [2000], and is used in FF.

Proposition 1 *Let $\mathcal{D}$ be a set of search spaces with heuristics, such that no search space contains a local minimum, and med is an upper limit to the maximal exit distance. Say we have a search space $(S, E, G, s_0) \in \mathcal{D}$, with heuristic h. Let b be the maximal number of outgoing edges of any state. Then, started on (S, E, G, s_0), Enforced Hill-climbing will find a goal state after considering $O(h(s_0) * b^{med+1})$ states.*

Without local minima, each iteration of Enforced Hill-climbing crosses a bench-related region or a contour, so it finds a better state at maximal depth $med + 1$, considering $O(b^{med+1})$ states. Each iteration improves the heuristic value by at least one, so after at most $h(s_0)$ iterations, a goal state is reached.

Reconsider the terminology introduced at the beginning of Section 3. Say we have a planning domain with operator schemata $\mathcal{O}$. Any task specifies, amongst other things, the set of objects O, yielding the action set A. An obvious upper limit to the number of outgoing edges in the task's state space is $|A|$. Furthermore, if the longest add list of any action has size k, then, for non dead end states s, $h^+(s) \leq k * |A|$. This is because with empty delete lists, each atom needs to be added at most once. Finally, $|A|$ and k are polynomial in $|O|$ for fixed $\mathcal{O}$. Thus, considering only the solvable tasks from a domain, applying Proposition 1 gets us the following. If h^+ does not yield any local minima, and produces a constant maximal exit distance, then Enforced Hill-climbing, using h^+, finds a goal state to each task by looking at a number of states polynomial in $|O|$.[1]

7 Explaining FF's Runtime Behavior

To give an example of how our results under evaluation with h^+ carry over to existing approximation techniques, we ran the same experiments, using the FF heuristic. The results are summarized in Figure 5.

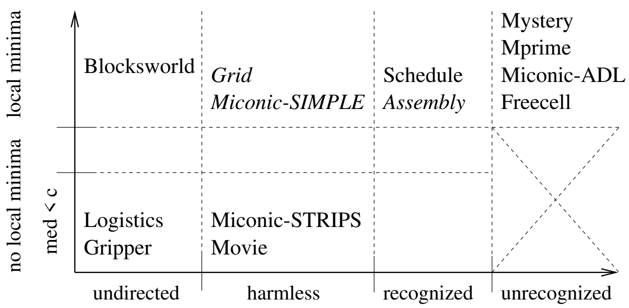

Figure 5: Results overview for the FF heuristic.

Comparing Figure 5 with Figure 4, one sees that there are three domains where local minima arise, when using the FF heuristic instead of h^+. However, the averaged valley percentage is below 6% in *Grid*, below 2% in *Assembly*, and below 0.3% in *Miconic-SIMPLE*. Four of the domains that are simple with h^+ stay simple with the FF heuristic.

As Hoffmann [2000] describes, the FF system uses the Enforced Hill-climbing algorithm as its search method. For STRIPS planning tasks, it is easy to see that, like it is the case for the h^+ heuristic, FF's heuristic estimate to any state is bounded by the number of actions. Thus, if Hypotheses 1 and 2 are true for the STRIPS domains *Gripper*, *Logistics*, *Miconic-STRIPS*, and *Movie*, under evaluation with the FF heuristic, then Enforced Hill-climbing, using that heuristic, solves the tasks in each of these domains by evaluating polynomially many states.

8 Conclusion and Outlook

The intuition that many planning benchmarks are "simple" in some sense is not new to the planning community. What the author personally likes most about the presented work is that it provides a formal notion of what simplicity, in that context, might mean. We give empirical data supporting that many benchmarks are, in fact, simple in that formal sense. The work provides insights into fundamental structural differences between different planning domains, and offers explaining the success of FF—and possibly of other state-of-the-art heuristic planners—as utilizing the simplicity of the benchmarks.

The presented results are preliminary to the effect that observations are made on a collection of comparatively small planning tasks. The stated hypotheses must be verified. For the h^+ function, we are going to prove our hypotheses analytically. For the FF and HSP heuristic functions, we are going to take samples from the state spaces of larger tasks.

Practically, we see the benefits of our results in mainly three areas. Firstly—which is a line of work that we are currently exploring—one can try to recognize simple planning tasks automatically, and thereby predict the runtime behavior of FF or other heuristic planners. Secondly, knowing about the strengths and weaknesses of existing heuristic functions may help in designing better ones. Finally, a better understanding of the structural differences between planning domains may help in designing more challenging benchmarks.

[1]In domains where this holds, deciding plan existence is in NP: To any solvable task, there is a solution plan with at most $h^+(\mathcal{I}) * (med + 1)$ steps, i.e., a plan of polynomial length.

Acknowledgments

The author thanks Bernhard Nebel for discussions on the presented work in all stages of its development. Thanks also go to Malte Helmert for many useful comments on an early version of the paper.

References

[Bacchus and Nau, 2001] Fahiem Bacchus and Dana Nau. The 2000 AI planning systems competition. *The AI Magazine*, 2001. Forthcoming.

[Bonet and Geffner, 2001] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 2001. Forthcoming.

[Bonet *et al.*, 1997] Blai Bonet, Gábor Loerincs, and Héctor Geffner. A robust and fast action selection mechanism for planning. In *Proc. AAAI-97*, pages 714–719. MIT Press, July 1997.

[Bylander, 1994] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.

[Edelkamp, 2000] S. Edelkamp. Heuristic search planning with BDDs. In *ECAI-Workshop: PuK*, 2000.

[Frank *et al.*, 1997] Jeremy Frank, Peter Cheeseman, and John Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.

[Gazen and Knoblock, 1997] B. Cenk Gazen and Craig Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *Proc. ECP'97*, pages 221–233. Springer-Verlag, September 1997.

[Hoffmann, 2000] Jörg Hoffmann. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proc. ISMIS-00*, pages 216–227. Springer-Verlag, October 2000.

[Koehler and Hoffmann, 2000] Jana Koehler and Jörg Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, 12:338–386, 2000.

[McDermott, 2000] Drew McDermott. The 1998 AI planning systems competition. *The AI Magazine*, 21(2):35–55, 2000.