What Makes The Difference Between HSP and FF?

Jörg Hoffmann* and **Bernhard Nebel**
Institute for Computer Science
<last-name>@informatik.uni-freiburg.de

Abstract

The HSP and FF systems are state-of-the-art domain independent planners. FF can historically be seen as a successor of HSP. It is based on the same ideas like HSP, but differs from its predecessor in a number of details. FF outperforms HSP in many planning domains. We have carried out a large scale experiment where we ran all configurations of FF's new techniques on a sizeable set of planning tasks. We describe the experimental design, and present our findings. The results give a clear picture of what the most important reasons are for FF's performance advantage over HSP.

1 Introduction

The HSP domain independent planning system [Bonet and Geffner, 1998] has successfully participated in the AIPS-1998 planning systems competition [McDermott, 2000]. This success has initiated a number of research efforts, amongst others [Bonet and Geffner, 1999; Refanidis and Vlahavas, 1999] Hoffmann's FF system [Hoffmann, 2000]. Both the HSP and the FF system approach planning by heuristic search, and both use the same base paradigm, first proposed by Bonet et al. [1997], for deriving their heuristic functions. However, FF outperforms HSP on many planning domains btoh in terms of running time and in terms of solution length. This has for example been observed in the recent AIPS-2000 planning systems competition [Bacchus and Nau, 2001]. This raises the question, if FF is so closely related to HSP, then why does it perform so much better? What makes the difference between the two systems? In this paper, we describe an experiment that we made in order to answer that question.

FF, like HSP, performs search in the state space—the space of all reachable states—and estimates the difficulty of search states by ignoring negative interactions. FF differs from HSP in a number of important details:

1. FF computes its heuristic in a different way than HSP, taking account of positive interactions.
2. FF uses a different variant of Hill-climbing than HSP does.
3. FF employs a pruning technique selecting the most promising successors of any search node.

We have implemented experimental code where each of FF's new techniques is attached to a switch, making it possible to turn them on and off independently of each other. Using all configurations of techniques, we measured runtime and solution length performance on a large set of planning tasks. Afterwards, for each pair of configurations in all planning domains we used, we decided whether performance was significantly improved or degraded. In the following we describe the experimental design, and present our findings.

Section 2 gives a short summary of the background in domain independent planning and in HSP's and FF's algorithms. Section 3 describes our experimental design, Sections 4 and 5 present our findings concerning running time and solution length behavior respectively. Section 6 concludes.

2 Background

One of the most popular planning frameworks is the STRIPS formalism [Fikes and Nilsson, 1971], which is based on propositional logics. A planning task $\mathcal{P}$ in STRIPS is a triple $\mathcal{P} = (A, \mathcal{I}, \mathcal{G})$ where A is the set of actions, $\mathcal{I}$ is the initial state, and $\mathcal{G}$ is the goal condition. $\mathcal{I}$ and $\mathcal{G}$ are sets of atomic facts, and each action $o \in A$ is a triple $o = (\text{pre}(o), \text{add}(o), \text{del}(o))$ containing the action's precondition-, add-, and delete-lists, respectively, which are all sets of atoms. World states are also represented as sets of atoms. An action o is applicable in a state S if $\text{pre}(o) \subseteq S$. Then, the resulting state $Result(S, o) = (S \cup \text{add}(o)) \setminus \text{del}(o)$. A plan is a sequence of actions that, when successively applied to the initial state, yields a state S that fulfills the goal condition, $\mathcal{G} \subseteq S$.

HSP and FF both search in the state space—the set of all reachable states—starting at the initial state and terminating when a goal state is found. Search is guided

*Georges-Köhler-Allee, Geb. 52, 79110 Freiburg, Germany, Phone: +49 (761) 203-8229, Fax: +49 (761) 203-8222, WWW: http://www.informatik.uni-freiburg.de/ hoffmann

by a heuristic function h that is based on relaxing the planning task as follows. For a task $\mathcal{P} = (A, \mathcal{I}, \mathcal{G})$, the relaxed task to $\mathcal{P}$ is $\mathcal{P}' = (A', \mathcal{I}, \mathcal{G})$ where $A' = \{(\text{pre}, \text{add}, \emptyset) \mid (\text{pre}, \text{add}, \text{del}) \in A\}$. In words, the relaxation ignores negative interactions by assuming that all delete lists are empty. When HSP or FF consider some search state S, they both obtain $h(S)$ by estimating the solution length of the relaxed task $(A', S, \mathcal{G})$, i.e., the length of a relaxed action sequence achieving the goal (the length of a shortest such sequence would be an admissible heuristic, but is NP-hard to compute). While HSP's estimates are based on computing certain weight values for all facts—assuming all facts are achieved independently—FF computes an explicit solution to $(A', S, \mathcal{G})$, which can take account of positive interactions.

Both HSP and FF use variations of local search, in the hope to find a goal state fast. While the search mechanism in HSP is a common form of Hill-climbing where to each search state one best successor is picked at random [Bonet and Geffner, 1998], FF uses a so-called enforced form of Hill-climbing [Hoffmann, 2000]. There, facing an intermediate search state S, the mechanism performs complete breadth first search until it finds a state S' that is strictly better, i.e., $h(S') < h(S)$. HSP's search mechanism outputs as solution plan the sequence of all actions it has used on its way from the initial state to the goal. FF outputs the concatenation of all action sequences that breadth first search found as connection from the states S to S'.

One final difference between FF and HSP is that to any search state S, FF expands only a subset of the state's successors, namely those that are generated by the so-called helpful actions $H(S)$. Loosely speaking, those are applicable actions that have something in common with the relaxed solution that FF's heuristic function has computed to S [Hoffmann, 2000].

Planning tasks are grouped together in domains, which constitute a family of related tasks. For example in the *Blocksworld* domain, the planner faces a collection of blocks that are assembled in stacks on a table, and needs to rearrange the blocks. Planning tasks (instances) then define the initial and goal stacks of all blocks. In our experiments, we have looked at a collection of 20 domains often used in the planning community, including all domains from the AIPS-1998 and AIPS-2000 competitions: *Assembly*, two *Blocksworlds* (with and without robot arm), *Briefcaseworld*, *Bulldozer*, *Freecell*, *Fridge*, *Grid*, *Gripper*, *Hanoi*, *Logistics*, *Miconic*-ADL, *Miconic*-SIMPLE, *Miconic*-STRIPS, *Movie*, *Mprime*, *Mystery*, *Schedule*, *Tireworld*, and *Tsp*. Fifteen of these are STRIPS domains, the others are specified in the more expressive ADL language.

3 Experimental Design

As said, we have implemented experimental code where each of FF's new techniques is attached to a switch, turning the technique on or off. The eight different configurations of the switches yield eight different heuristic planners. With all switches off, we imitated HSP. Concerning the goal distance estimates switch and the pruning techniques switch, we implemented the original methods. Concerning the search strategy, we used the following simple Hill-climbing design:

- Always select one best evaluated successor randomly.
- Keep a memory of past states to avoid cycles in the Hill-climbing path.
- Count the number of consecutive times in which the child of a node does not improve the heuristic estimate. If that counter exceeds a threshold, then restart, where the threshold is 2 times the initial state's goal distance estimate.
- Keep visited nodes in memory across restart trials in order to avoid multiple computation of the heuristic for the same state.

In HSP, some more variations of restart techniques are implemented. In personal communication with Blai Bonet and Hector Geffner, we decided not to imitate those variations—which affect behavior only in a few special cases—and use the simplest possible design instead.

To obtain data, we set up a large example suite, containing a total of 939 planning tasks from our 20 domains. In *Hanoi*, there were 8 tasks—3 to 10 discs to be moved—in the other domains, we used from 30 to 69 different instances. As very small instances are likely to produce noisy data, we tried to avoid those by rejecting tasks that were solved by FF in less than 0.2 seconds. Instances were either taken from published distributions or randomly generated.

For each of the eight configurations of switches, we ran the respective planner on all tasks in our example suite. Those configurations using randomized Hill-climbing were run five times on each task, and the results averaged afterwards. To complete the experiments in a reasonable time, we restricted memory consumption to 128 MByte, and time consumption to 150 seconds. We examined the data separately for each domain, as our algorithmic techniques typically show similar behavior for all tasks within a domain.

4 Running Time

For our running time investigation, if a configuration did not find a solution plan to a given task, we set the respective running time value to the time limit of 150 seconds. In the following, we designate each switch configuration by 3 letters: "H" stands for helpful actions on, "E" stands for Enforced Hill-climbing on, "F" stands for FF estimates on. If a switch is turned off, the respective letter is replaced by a "$-$": FF is configuration "HEF", our HSP imitation is "$- - -$", and "H$--$", for example, is Hill-climbing with HSP goal distances and helpful actions pruning. For a first impression of our running time results, see the averaged values per domain in Figure 1.

Figure 1 shows, for each domain and each configuration, the averaged running time over all instances in that domain. As the instances in each domain are not all the same size, but typically scale from smaller to very large tasks, averaging over all running times is of course a very crude approximation of runtime behavior. The data in Figure 1 provides a general impression of our runtime results per domain, and gives a few hints on the phenomena that might be present in the data. Compare, for example, the values on the right hand side—those planners using helpful actions—to those on the left hand side—those planners expanding all sons of search nodes. In *Briefcaseworld* and *Bulldozer*, the right hand side values are higher, but in almost all other domains, they are considerably lower. This is especially true for the two rightmost columns, showing values for planners using helpful actions and Enforced Hill-climbing. This indicates that the main sources of performance lie in the pruning technique and the search strategy—looking at the rightmost "HE$-$" and "HEF" columns, which only differ in the goal distance estimate, those two configuration values are usually close to each other, compared to the other configurations in the same domain.

To put our observations on a solid basis, we looked, for each domain, at each pair of configurations in turn, amounting to $20 * \frac{8*7}{2} = 560$ pairs of planner performances. For each such pair, we decided whether one configuration performed significantly better than the other one. To decide significance, we counted the number of tasks that one configuration solved faster. We found this to be a more reliable criterion than things like the difference between running times for each task. As tasks grow in size, rather than being taken from a population with finite mean size, parametric statistical procedures like computing significance intervals for runtime differences make questionable assumptions about the distribution of data. We thus used the following non-parametric statistical test, known as the *two-tailed sign test* [Siegel and Castellan, 1988]. Assume that both planners, A and B, perform equally on a given domain. Then, given a random instance from the domain, the probability that B is faster than A should be equal to the probability that A is faster than B. Take this as the null hypothesis. Under that hypothesis, if A and B behave differently on an instance, then B is faster than A with probability $\frac{1}{2}$. Thus, the tasks where B is faster are distributed over the tasks with different behavior according to a Binomial distribution with $p = \frac{1}{2}$. Compute the probability of the observed outcome under the null hypothesis, i.e., if there are n tasks where A and B behave differently, and k tasks where B is faster, then compute the probability that, according to a binomial distribution with $p = \frac{1}{2}$, at least k positive outcomes are obtained in n trials. If that probability is less or equal than .01, then reject the null hypothesis and say that B performs significantly better than A. Symmetrically, decide whether A performs significantly better than B. We remark that in all domains except *Movie* the tasks where two configurations behaved equally were exactly those that could not be solved by either of the configurations. In 60% of the cases where we found that one configuration B performed significantly better than another configuration, B was faster on *all* instances with different behavior. In 71%, B was faster on all but one such instance.

We are particularly interested in pairs A and B of configurations where B results from A by turning one of the switches on, leaving the two others unchanged. Deciding about significant improvement in such cases tells us about the effect that the respective technique has on performance in a domain. There are 12 pairs of configurations where one switch is turned on. Figure 2 shows our findings in these cases.

Figure 2 is to be understood as follows. It shows our results for the "F", "E", and "H" switches, which become active in turn from left to right. For each of these switches, there are four configurations of the two other, background, switches, displayed by four columns in the table. In each column, the behavior of the respective background configuration with the active switch turned off is compared to the behavior with the active switch turned on. If performance is improved significantly, the table shows a "$+$", if it is significantly degraded, the table shows a "$-$", and otherwise the respective table entry is empty. For example, consider the top left corner, where the "F" switch is active, and the background configuration is "$--$", i.e., Hill-climbing without helpful actions. Planner A is "$---$", using HSP distances, and planner B is "$--$F", using FF distances. B's performance is significantly better than A's, indicated by a "$+$".

The leftmost four columns in Figure 2 show our results for HSP distance estimates versus FF distance estimates. Clearly, the latter estimates are superior in our domains, in the sense that for each background configuration the behavior gets significantly improved in 9 to 12 domains. In contrast, there are only 5 cases altogether where performance gets worse. The significances are quite scattered over the domains and background configurations, indicating that a lot of the significances result from interactions between the techniques that occur only in the context of certain domains. For example, performance is improved in *Bulldozer* when the background configuration does not use helpful actions, but degraded when the background configuration uses Hill-climbing with helpful actions. This kind of behavior can not be observed in any other domain. There are 4 domains where performance is improved in all but one background configuration. Apparently in these cases some interaction between the techniques occurs only in one specific configuration. We remark that often running times with FF's estimates are only a little better than with HSP's estimates, i.e., behavior gets improved reliably over all instances,

	− − −	− −F	−E−	−EF	H− −	H−F	HE−	HEF
Assembly	117.39	31.75	92.95	61.10	47.81	20.25	20.34	16.94
Blocksworld	4.06	2.53	8.37	30.11	1.41	0.83	0.27	6.11
Blocksworld-arm	0.60	8.81	80.02	56.20	1.21	10.13	25.19	40.65
Briefcaseworld	16.35	5.84	66.51	116.24	150.00	150.00	150.00	150.00
Bulldozer	4.47	3.24	31.02	15.74	81.90	126.50	128.40	141.04
Freecell	65.73	46.05	54.15	51.27	57.35	42.68	43.99	41.44
Fridge	28.52	53.58	31.89	52.60	0.85	0.69	1.88	2.77
Grid	138.06	119.53	115.05	99.18	115.00	95.10	18.73	11.73
Gripper	2.75	1.21	15.16	1.00	1.17	0.48	0.17	0.11
Hanoi	93.76	75.05	6.29	3.91	150.00	78.82	4.47	2.70
Logistics	79.27	102.09	79.77	111.47	36.88	39.69	10.18	11.94
Miconic-ADL	150.00	150.00	102.54	54.23	142.51	128.28	95.45	59.00
Miconic-SIMPLE	2.61	2.01	2.47	1.93	1.35	0.86	0.55	0.56
Miconic-STRIPS	2.71	2.32	4.84	1.53	1.44	1.01	0.64	0.36
Movie	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Mprime	73.09	69.27	82.89	81.43	47.09	58.45	18.56	26.62
Mystery	78.54	90.55	71.60	86.01	75.73	95.24	85.13	86.21
Schedule	135.50	131.12	143.59	141.42	77.58	38.23	12.23	13.77
Tireworld	135.30	110.38	119.22	121.34	121.13	105.67	97.41	85.64
Tsp	4.11	0.82	2.45	0.75	2.48	0.57	0.15	0.07

Figure 1: Averaged running time per domain for all eight configurations of switches.

but only by a small factor (to get an idea of that, compare the differences between average running times in Figure 1, for configurations where only the distance estimate changes). In 5 domains, FF's estimates improve performance consistently over all background configurations, indicating a real advantage of the different distance estimates.

Comparing Hill-climbing versus Enforced Hill-climbing, i.e., looking at the four columns in the middle of Figure 2, the observation is this. The different search technique is a bit questionable when the background configuration does not use helpful actions, but otherwise Enforced Hill-climbing yields excellent results: without helpful actions performance gets degraded almost as many times as it gets improved whereas with helpful actions Enforced Hill-climbing improves performance significantly in 16 of our 20 domains, being degraded only in *Fridge*. We draw two conclusions. First, whether one or the other search strategy is adequate depends very much on the domain. A simple example for that is the *Hanoi* domain, where Hill-climbing always restarts before it can reach the goal—on all paths to the goal, there are exponentially many state transitions where the son has no better evaluation than the father. Second, there is an interaction between Enforced Hill-climbing and helpful actions pruning that occurs consistently across almost all of our planning domains. This can be explained by the effect that the pruning technique has on the different search strategies. In Hill-climbing, helpful actions pruning prevents the planner from looking at too many superfluous successors on each single state that a path goes through. This saves time proportional to the length of the path. The effects on Enforced Hill-climbing are much more drastic. There, helpful actions prunes out unnecessary successors of each state during a breadth first search, i.e., it cuts down the branching factor, yielding performance speedups exponential in the depths that are encountered.

We finally compare consideration of all actions versus consideration of only the helpful ones. Look at the rightmost four columns of Figure 2. The observation is simply that helpful actions are really helpful—they improve performance significantly in almost all of our planning domains. This is especially true for those background configurations using Enforced Hill-climbing, due to the same interaction that we have outlined above. In some domains, helpful actions pruning imposes a very rigid restriction on the search space: in *Schedule* we found that states can have hundreds of successors, where only about 2% of those are considered helpful. In other domains only a few actions are pruned, like in *Hanoi* where at most three actions are applicable in each state which are all considered helpful in most of the cases. Even a small degree of restriction does usually lead to a significant improvement in performance. In two domains, *Briefcaseworld* and *Bulldozer*, helpful actions can prune out too many possibilities, i.e., they cut away solution paths. This happens because there the relaxed plan can ignore things that are crucial for solving the real task.

5 Solution Length

We also investigated the effects that FF's new techniques have on solution length. Comparing two configurations A and B, we took as the data set the respective solution length for those tasks that both A and B managed to solve—obviously there is not much point in comparing solution length when one planner can not

	F				E				H			
domain	−−	−E	H−	HE	−−	−F	H−	HF	−−	−F	E−	EF
Assembly	+	+	+	+		−	+	+	+	+	+	+
Blocksworld	+		+	−	+	−	+	+	+	+	+	+
Blocksworld-arm		+			−	−	−			−	+	+
Briefcaseworld	+	−			−	−			−	−	−	−
Bulldozer	+	+	−		−	−	−		−	−	−	−
Freecell	+	+	+	+	+	+	+	+	+	+	+	+
Fridge	−				−	−	−	−	+	+	+	+
Grid	+		+	+	+		+	+	+	+	+	+
Gripper	+	+	+	+	−	+	+	+	+	+	+	+
Hanoi		+			+	+	+	+				
Logistics	−	−	+			−	+	+	+	+	+	+
Miconic-ADL		+		+	+	+	+	+		+	+	+
Miconic-SIMPLE	+	+	+		+	+	+	+	+	+	+	+
Miconic-STRIPS	+	+	+	+		+	+	+	+	+	+	+
Movie					+	+	+	+				
Mprime	+	+	+				+	+	+	+	+	+
Mystery					+		+	+	+			
Schedule			+	+		−	+	+	+	+	+	+
Tireworld	+		+	+		−	+	+		+	+	+
Tsp	+	+	+	+	+	+	+	+	+	+	+	+

Figure 2: The effect of turning on a single switch, keeping the others unchanged. Summarized in terms of significantly improved or degraded running time performance per domain, and per switch configurations.

find a solution at all. We then counted the number n of tasks where A and B behaved differently, and the number k where B's solution was shorter, and decided about significance like described in the last section. Figure 3 shows our results in those cases where a single switch is turned.

The data in Figure 3 is organized in the obvious manner analogous to Figure 2. A first glance at the table tells us that FF's new techniques are also useful for shortening solution length in comparison to HSP, but not as useful as they are for improving runtime behavior. Let us focus on the leftmost four columns, HSP distance estimates versus FF distance estimates. The observations are that, with Enforced Hill-climbing in the background, FF estimates often result in shorter plans, and that there are two domains where solution lengths are improved across all background configurations. Concerning the second observation, having a closer look at those two domains we found that there FF's heuristic recognizes properties of the domain that HSP's heuristic doesn't notice. Concerning the first observation, improved solution lengths when Enforced Hill-climbing is in the background, we do not have a good explanation for this. It seems that the greedy way in which Enforced Hill-climbing builds its plans is just better suited when distance estimates are cautious, i.e., low.

Consider the four columns in the middle of Figure 3, Hill-climbing versus Enforced Hill-climbing. There are many cases where the different search strategy results in shorter plans. We figure that this is due to the different plateau behavior that the search methods exhibit, i.e., their behavior in flat regions of the search space. Enforced Hill-climbing enters a plateau somewhere, performs complete search for a state with better evaluation, and adds the shortest path to that state to its current plan prefix. When Hill-climbing enters a plateau, it strolls around more or less randomly until it hits a state with better evaluation or has enough of it and restarts. All the actions on its journey to the better state are kept in the final plan.

Finally, we compare consideration of all actions versus consideration of only the helpful ones, results depicted in the rightmost four columns of Figure 2. Coming a bit unexpected, there is only one single case where solution length performance is degraded by turning on helpful actions. This indicates that the actions on the shortest path to the goal are, in fact, usually considered helpful—unless *all* solution paths are thrown away, as is sometimes the case only in the *Briefcaseworld* and *Bulldozer* domains. Pruning the search space with helpful actions sometimes even leads to significantly shorter solution plans, especially when the underlying search method is Hill-climbing. Though this may sound paradoxical, there is a simple explanation to it. Consider what we said above about the plateau behavior of Hill-climbing, randomly adding actions to the current plan in the search for a better state. If such a search engine is armed with the helpful actions successors choice, focusing it into the direction of the goals, it might well take less steps to find the way off a plateau.

6 Conclusion and (some) Outlook

We have carried out a large-scale experiment comparing different configurations of FF with its prede-

	F				E				H			
domain	−−	−E	H−	HE	−−	−F	H−	HF	−−	−F	E−	EF
Assembly		+	+	+		+	+		+	+		
Blocksworld		+		+		+		+		+	+	
Blocksworld-arm				+	+	+	+	+	+	+	+	
Briefcaseworld		+			+	+						
Bulldozer					+	+						
Freecell		+		+					+	+		
Fridge	−		−		+	+		+	+	+		
Grid				+	+	+	+	+	+	+		
Gripper	+	+	+	+	+		−				−	
Hanoi												
Logistics		+	−	+	+	+	+	+	+		+	
Miconic-ADL		+		+				+				+
Miconic-SIMPLE	−	+	+	+		+	+	+		+	+	+
Miconic-STRIPS	+	+	+	+	+	+	+	+			+	+
Movie					−	−	−	−				
Mprime												
Mystery												
Schedule			+					−	+	+		
Tireworld				+				+				
Tsp												

Figure 3: The effect of turning on a single switch, keeping the others unchanged. Summarized in terms of significantly improved or degraded solution length performance per domain, and per switch configurations.

cessor HSP. The results show that the main sources of improved runtime performance in FF are the helpful actions pruning technique and the Enforced Hill-climbing search strategy, as well as their interaction. Concerning improved solution length performance, the reason for that is mainly the new search strategy and its (yet unexplained) interplay with the different distance estimates. We believe that our experiment has also value for the planning community in that it gives an example of a large scale empirical study on comparative planner performance. Studies like that are regrettably rare in the planning literature.

One idea for future research arises from special cases that occured in our experiments. An example is *Fridge*. While in most other domains Enforced Hill-climbing with helpful actions pruning is most adequate, in this domain simple Hill-climbing has clear runtime advantages. Other unusual examples are *Briefcaseworld* and *Bulldozer* where helpful actions cuts out the solutions, or *Logistics* where HSP's estimates sometimes lead to better runtime performance. The challenge is to find ways of automatically recognizing such special cases, in order to configure the most appropriate solving technology.

References

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

[Bonet and Geffner, 1998] Blai Bonet and Héctor Geffner. HSP: Heuristic search planner. In *AIPS-98 Planning Competition*, Pittsburgh, PA, 1998.

[Bonet and Geffner, 1999] Blai Bonet and Héctor Geffner. Planning as heuristic search: New results. In *Proc. ECP-99*. Springer-Verlag, September 1999.

[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.

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[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.

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

[Refanidis and Vlahavas, 1999] Ioannis Refanidis and Ioannis Vlahavas. GRT: a domain independent heuristic for STRIPS worlds based on greedy regression tables. In *Proc. ECP-99*. Springer-Verlag, September 1999.

[Siegel and Castellan, 1988] S. Siegel and Jr. N. J. Castellan. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill, 2nd edition, 1988.