ECAI 2008
M. Ghallab et al. (Eds.)
IOS Press, 2008

doi:10.3233/978-1-58603-891-5-588

Heuristics for Planning with Action Costs Revisited

Emil Keyder[1] and **Héctor Geffner**[2]

Abstract. We introduce a simple variation of the additive heuristic used in the HSP planner that combines the benefits of the original additive heuristic, namely its mathematical formulation and its ability to handle non-uniform action costs, with the benefits of the relaxed planning graph heuristic used in FF, namely its compatibility with the highly effective enforced hill climbing search along with its ability to identify helpful actions. We implement a planner similar to FF except that it uses relaxed plans obtained from the additive heuristic rather than those obtained from the relaxed planning graph. We then evaluate the resulting planner in problems where action costs are not uniform and plans with smaller overall cost (as opposed to length) are preferred, where it is shown to compare well with cost-sensitive planners such as SGPlan, Sapa, and LPG. We also consider a further variation of the additive heuristic, where symbolic labels representing action sets are propagated rather than numbers, and show that this scheme can be further developed to construct heuristics that can take delete-information into account.

1 PLANNING MODEL AND HEURISTICS

We consider planning problems $P = \langle F, I, O, G\rangle$ expressed in Strips, where F is the set of relevant atoms or fluents, $I \subseteq F$ and $G \subseteq F$ are the initial and goal situations, and O is a set of (grounded) actions a with precondition, add, and delete lists $Pre(a)$, $Add(a)$, and $Del(a)$ respectively, all of which are subsets of F. For each action $a \in O$, we assume that there is a *non-negative* $cost(a)$ so that the cost of a plan $\pi = a_1, \ldots, a_n$ is

$$cost(\pi) = \sum_{i=1}^{n} cost(a_i) \tag{1}$$

This cost model is a generalization of the classical model where the cost of a plan is given by its length. Two of the heuristics used to guide the search for plans in the classical setting are the *additive heuristic* h_a used in HSP [2], and the *relaxed plan heuristic* h_{FF} used in FF [11]. Both are based on the delete relaxation P^+ of the problem, and both attempt to approximate the optimal delete-relaxation heuristic h^+ which is well-informed but intractable. We review these heuristics below. In order to simplify the definition of some of the heuristics, we introduce a new dummy End action with *zero cost*, whose preconditions $G_1, \ldots, G_n$ are the goals of the problem, and whose effect is a dummy atom G. The heuristics $h(s)$ then estimate the cost of achieving this 'dummy' goal G from s.

[1] Universitat Pompeu Fabra, Passeig de Circumvalació 8, 08003 Barcelona, Spain. email: emil.keyder@upf.edu
[2] ICREA & Universitat Pompeu Fabra, Passeig de Circumvalació 8, 08003 Barcelona, Spain. email: hector.geffner@upf.edu

1.1 The Additive Heuristic

Since the computation of the optimal delete-free heuristic h^+ is intractable, HSP introduces a polynomial approximation in which subgoals are assumed to be *independent* in the sense that they are achieved with no 'side effects' [2]. This assumption is normally false, but results in a simple heuristic function

$$h_a(s) \stackrel{\text{def}}{=} h(G; s) \tag{2}$$

that can be computed quite efficiently in every state s visited in the search from the recursive equation:

$$h(p; s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s \\ h(a_p; s) & \text{otherwise} \end{cases} \tag{3}$$

where $h(p; s)$ stands for an estimate of the cost of achieving the atom p from s, $h(a; s)$ stands for an estimate of the cost of applying action a in s, and a_p is a *best support* of fluent p in s. These two expressions are defined in turn as

$$h(a; s) \stackrel{\text{def}}{=} cost(a) + \sum_{q \in Pre(a)} h(q; s) \tag{4}$$

and

$$a_p \stackrel{\text{def}}{=} \text{argmin}_{a \in O(p)} h(a; s) \tag{5}$$

where $O(p)$ stands for the actions in the problem that add p. Versions of the additive heuristic appear also in [6, 16, 17], where the cost of joint conditions in action preconditions or goals is set to the sum of the costs of each condition in isolation. When the 'sum' in (4) is replaced by 'max', the heuristic h_{max} is obtained [2]. The heuristic h_{max}, unlike the additive heuristic h_a, is admissible, but less informed. The heuristics coincide and are equivalent to the optimal delete-relaxation heuristic h^+ when all the actions involve a single precondition and the goal involves a single atom.

1.2 The Relaxed Planning Graph Heuristic

The planner FF modifies HSP along two dimensions: the heuristic and the search algorithm. Unlike h_a, the heuristic h_{FF} used in FF makes no independence assumption for approximating h^+, computing instead one plan for P^+ which is not guaranteed to be optimal. This is done by a Graphplan-like procedure [1], which due to the absence of deletes constructs a planning graph with no mutexes, from which a plan $\pi_{\text{FF}}(s)$ is extracted backtrack-free [11]. The heuristic $h_{\text{FF}}(s)$ is then set to $|\pi_{\text{FF}}(s)|$. The basic search procedure in FF is not *best-first* as in HSP but (enforced) *hill-climbing* (EHC), in which the search moves from a state s to a neighboring state s' with smaller heuristic value by performing a *breadth first search.* This breadth first search is carried out with a *reduced branching factor*, ignoring actions a that are not found to be 'helpful'. The 'helpful actions' in

a state s are the actions applicable in s that add the precondition p of an action in $\pi_{\text{FF}}(s)$ for $p \notin s$. The use of EHC search, along with the pruning of non-helpful actions are the key factors that make FF scale up better than HSP in general [11], but due to its construction, the heuristic h_{FF} cannot be extended easily to take action costs into account (yet see [7]).

1.3 Relaxed Plans without Planning Graphs

A simple variation of the additive heuristic can be defined that is cost sensitive and results in relaxed plans compatible with helpful action pruning and EHC search. For this, the best support a_p of each atom p in the state s, calculated as part of the computation of the heuristic $h_a(s)$ in Equation 5, is stored.[3] The definition of the *set of actions* $\pi_a(s)$ that make up a relaxed plan then simply collects these best supports backwards from the goal:

$$\pi_a(s) \stackrel{\text{def}}{=} \pi(G; s)$$

$$\pi(p; s) \stackrel{\text{def}}{=} \begin{cases} \{\} & \text{if } p \in s \\ \{a_p\} \cup \bigcup_{q \in pre(a_p)} \pi(q; s) & \text{otherwise} \end{cases}$$

Intuitively, the relaxed plan $\pi_a(p; s)$ is empty if $p \in s$, and the union of the best supporter a_p for p with the relaxed plans for each of its preconditions $q \in pre(a_p)$ otherwise. Note that $\pi_a(s)$, being a set of actions, can contain an action at most once. The same construction, captured by Equation 6, underlies the construction of the relaxed plan $\pi_{\text{FF}}(s)$ computed by FF from the relaxed planning graph. For this, however, the best supports a_p that encode the 'best' actions for achieving the atom p in the relaxation, must be obtained from the h_{max} heuristic and not from h_a; a modification that just involves changing the sum operator in (4) by the max operator. The h_{max} heuristic is known to encode the first level of the relaxed planning graph that contains a given action or fact.

It is simple to prove that the collection of actions in $\pi_a(s)$ represents a plan from s in the delete relaxation P^+. This relaxed plan, unlike the relaxed plan $\pi_{\text{FF}}(s)$ is sensitive to action costs, and can be used in FF in place of $\pi_{\text{FF}}(s)$. We call the resulting planner FF(h_a).

2 THE FF(h_a) PLANNER

In FF(h_a), the relaxed plans $\pi_a(s)$ are produced by computing the additive heuristic using a Bellman-Ford algorithm while keeping track of the chosen lowest-cost supporter for each atom, and then recursively collecting the best supporters starting from the goal. The heuristic $h(s)$ used for measuring progress in FF(h_a) is defined as the relaxed plan *cost* $\sum_{a \in \pi_a(s)} cost(a)$ and not as its *length* $|\pi_a(s)|$. This heuristic, which is obtained from the computation of the additive heuristic h_a, is almost equivalent to $h_a(s)$ but does not count the cost of an action more than once.

The EHC search used in FF(h_a) is a slightly modified version of that used in FF. While a single step of EHC in FF ends as soon as a state s' is found by breadth-first search from s such that $h(s') < h(s)$, in FF(h_a), all states s' resulting from applying a helpful action a in s are evaluated and among those for which $h(s') < h(s)$ holds, the action minimizing the expression $cost(a) + h(s')$ is selected. Like in FF, the helpful actions in s are the actions applicable in s that add the precondition p of an action in $\pi_a(s)$ such that $p \notin s$.

[3] We assume that ties in the selection of the best supports a_p are broken arbitrarily. The way ties are broken does not affect the value of the additive heuristic $h_a(s)$ in a state s but may affect the value of the heuristic defined below. The same is true for FF's heuristic.

FF(h_a) is implemented on top of the Metric-FF planner [10] because of its ability to handle numeric fluents, through which non-uniform action costs are currently expressed in PDDL. *FF(h_a) does not make use of numeric fluents for any other purpose besides representing action costs.*

3 EXPERIMENTAL RESULTS

We evaluated the performance of FF(h_a) in comparison with other cost-sensitive planners; namely SGPlan [5], LPG-quality [8] and Sapa [6][4] on 11 domains.[5] For reference, the curves show also the plan times and costs obtained by running FF, that ignores cost information, and FF-quality, an option in Metric-FF that optimizes a given plan metric by using an FF-like heuristic in a Weighted A* search [10].

Experiments were performed with eleven domains, five of these taken from the numeric track of the Third International Planning Competition (IPC3). Of these 5 domains, the *Depots*, *Rovers*, *Satellite*, and *Zenotravel* domains were modified by *removing all occurrences of numeric variables from action preconditions and goals, once the action cost information was extracted from the PDDL.* Also, as a reference, all planners except LPG were evaluated on the STRIPS (uniform cost) versions of these domains, and all planners were evaluated on 6 new domains introduced here, which were constructed with the aim that the length of solutions not correlate with their cost. Indeed, in two of these domains, the *Minimum Spanning Tree* and *Assignment* domains, all valid solutions contain the same number of actions. The other domains are: *Shortest Path* (shortest-path problems), *Colored Blockworld* (blocks have colors and colors must be stacked in certain ways in the goal, with costs associated with the different blocks), *Delivery* (a variation of the IPC5 domain TPP), and *Simplified Rovers* (a domain adapted from [17], in which a robot must collect samples from rocks in a grid). Moreover, for *S. Rovers*, both hard goal and soft goal versions were used, with the soft goals being compiled away into action costs, following the procedure described in [13].[6] The experiments were run on a CPU running at 2.33 GHz with 8 GB of RAM. Execution time was limited to 1,800 seconds. The results, including plan costs and planning times for the various planners, are reported in the figures. Some observations about the results follow.

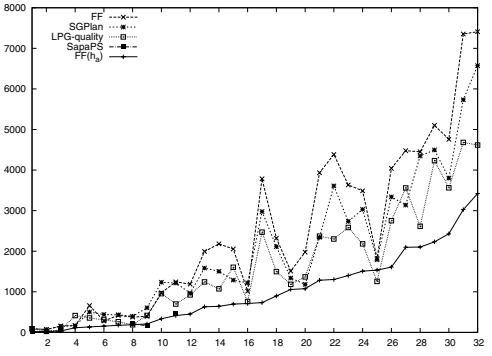
Quality of Plans: In almost all of the domains, FF(h_a) produces the best plans, with the exception of the hard-goal version of *S. Rovers* (Fig. 3c), where it does particularly bad, and in the soft version (Fig. 3b). In both cases, LPG does better, although the opposite occurs in several domains like in *Delivery* (Fig. 1a), *Satellite* (Fig. 2a), and the *Assignment Problem* (Fig 2c). Sapa produces plans that are close to the best quality plans in all the domains for which it can be executed, yet is usually able to solve only the smallest instances in each domain. FF-quality suffers from a similar problem, solving a significant proportion of the instances in a few domains only.[7] Overall, SGPlan does not appear to produce better plans than FF, even if FF ignores costs completely, and both produce plans that are often much worse than FF(h_a). In the STRIPS versions of the

[4] Sapa was compiled from Java to native machine code with the GNU compiler. We were later informed by the authors that this results in a slowdown of approximately 50% compared to the version running on the Java virtual machine.

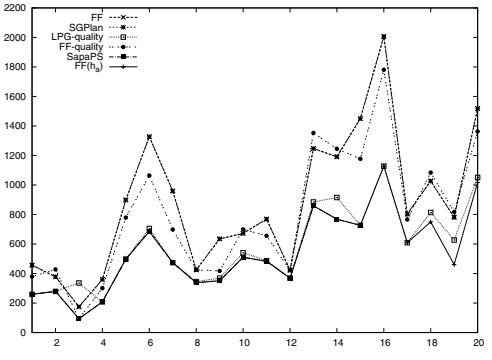
[5] LPG and Sapa could not be run on some of the domains due to bugs.

[6] We cannot provide further details on these domains due to lack of space, but the PDDL files are available from the authors.

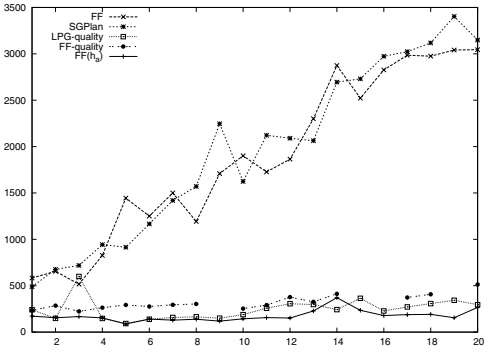
[7] For clarity, FF-quality's results are shown only for domains in which it was able to solve a significant number of instances.



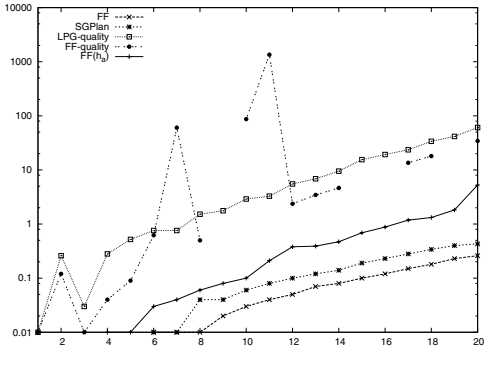
(a) Plan costs - *Delivery* domain



(b) Plan costs - *Shortest Path* domain

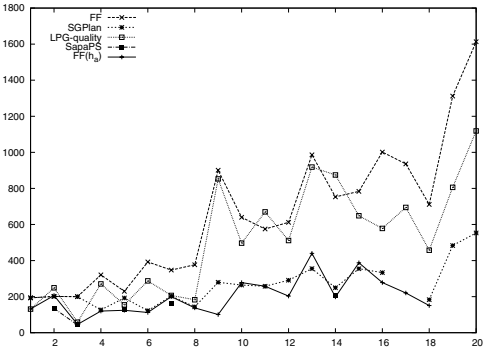


(c) Plan costs - *Minimum Spanning Tree* domain

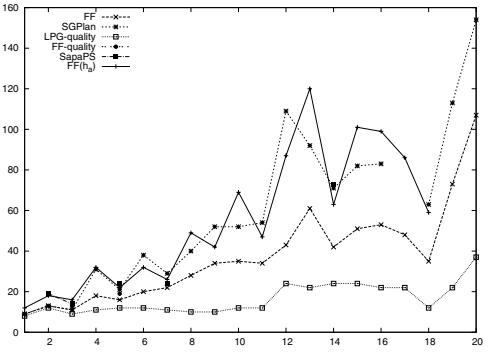


(d) Planning times - *Minimum Spanning Tree* domain

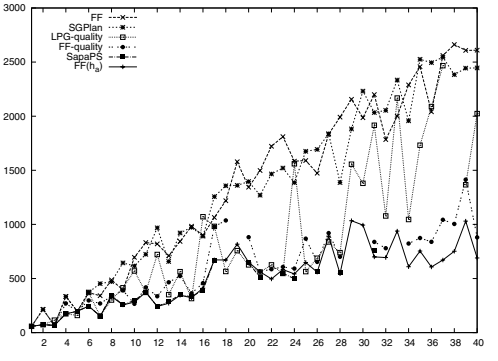
Figure 1.



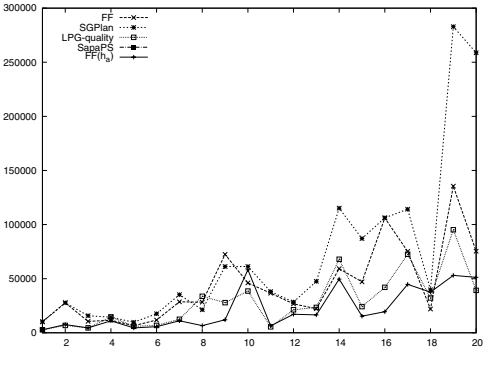
(a) Plan costs - *Satellites* domain



(b) Length of Plans above in *Satellites*



(c) Plan costs - *Assignment Problem* domain



(d) Plan costs - *Zenotravel* domain

Figure 2.

five IPC3 domains (unit costs), all planners produce plans of roughly equal quality.

Planning Times: FF(h_a) is somewhat slower than FF on most problems, though the difference is usually a constant factor (Fig. 1d).[8] There are two main reasons for this. The first is that computing h_a and extracting the associated relaxed plan π_a is somewhat more costly than the equivalent operation on the relaxed planning graph, so FF(h_a) takes longer to perform the same number of heuristic evaluations as FF. In general, h_{FF} evaluates states 2–10 times faster than h_a. The second is that while FF minimizes the number of actions in the plan, FF(h_a) minimizes the cost of the plan, which in some cases leads to longer plans, requiring more search nodes and more heuristic evaluations (Fig. 2b). SGPlan takes roughly the same amount of time as FF on almost all domains considered, while LPG is roughly an order of magnitude slower than the other planners except Sapa, but appears to have better scaling behaviour. Sapa is slower than LPG by roughly one order of magnitude.

4 FURTHER VARIATIONS OF THE ADDITIVE HEURISTIC

We consider briefly two further variations of the additive heuristic: the *set-additive* heuristic and the *TSP heuristic*, both analyzed in more detail in [12, 13].

4.1 The Set Additive Heuristic

In the additive heuristic, the value $h(a_p; s)$ of the best supporter a_p of p in s is propagated to obtain the heuristic value $h(p; s)$ of p. In contrast, in the *set-additive* heuristic, the best supporter a_p of p is itself propagated, with supports combined by *set-union* rather than by *sum*, resulting in a recursive function $\pi(p; s)$ that represents the *set of actions* in a relaxed plan for p in s, which can be defined similarly to $h(p; s)$ as:

$$\pi(p;s) = \begin{cases} \{\} & \text{if } p \in s \\ \pi(a_p;s) & \text{otherwise} \end{cases} \quad (6)$$

where

$$a_p = \mathrm{argmin}_{a \in O(p)} Cost(\pi(a;s)) \quad (7)$$

$$\pi(a;s) = \{a\} \bigcup \{\cup_{q \in Pre(a)} \pi(q;s)\} \quad (8)$$

$$Cost(\pi(a;s)) = \sum_{a' \in \pi(a;s)} cost(a') \quad (9)$$

The *set-additive heuristic* $h^s_a(s)$ for a state s is then defined as

$$h^s_a(s) = Cost(\pi(G;s)) \; . \quad (10)$$

It is easy to show that the collection of actions $\pi(p; s)$ for all atoms p represent plans for achieving the atom p in the delete-relaxation P^+, which in the set-additive heuristic are computed recursively, starting with the trivial (empty) plan for the atoms $p \in s$. From a practical point of view, this recursive computation does not appear to be cost-effective in general, as the relaxed plans $\pi_a(p; s)$ obtained from the normal additive heuristic are normally as good and can be computed faster. Yet the planner FF(h^s_a) obtained from FF by replacing the relaxed plans $\pi_{\mathrm{FF}}(s)$ by $\pi(G; s)$ above compares well with existing cost-sensitive planners (see [12]), and the formulation of the set-additive heuristic opens the door to the formulation of a broader family of heuristics.

[8] We omit further data on planning time due to space considerations.

4.2 The TSP Heuristic

The set-additive heuristic can be generalized by replacing the plans $\pi(p; s)$ with more generic **labels** $L(p; s)$ that can be numeric, symbolic, or a suitable combination, provided that there is a function $Cost(L(p; s))$ mapping labels $L(p; s)$ to numbers. Here we consider labels $L(p; s)$ that result from treating one designated multivalued variable X in the problem in a special way. A multivalued variable X is a set of atoms $x_1, \ldots, x_n$ such that exactly one x_i holds in every reachable state. For example, in a task where there are n rocks r_1, $\ldots$, r_n to be picked up at locations $l_1, \ldots, l_n$, the set of atoms $at(l_0)$, $at(l_1)$, $\ldots$, $at(l_n)$, where $at(l_0)$ is the initial agent location, represent one such variable, encoding the possible locations of the agent. If the cost of going from location l_i to location l_k is $c(l_i, l_k)$, then the cost of picking up all the rocks is the cost of the best (min cost) *path* that visits all the locations, added to the costs of the pickups. This problem is a TSP and therefore intractable, but its cost can be approximated by various fast suboptimal TSP algorithms.[9] By comparison, the delete relaxation approximates the cost of the problem as the cost of the best *tree* rooted at l_0 that spans all of the locations. The modification of the labels $\pi(p; s)$ in the set-additive heuristic allows us to move from the *approximate model* captured by the delete relaxation to *approximate TSP algorithms* over a more accurate model (see [15] for other uses of OR models in planning heuristics).

For this, we assume that the actions that affect the selected multivalued variable X do not affect other variables in the problem, and maintain in each label $\pi(p; s)$ two disjoint sets: a set of *actions* that do not affect X, and the set of *X-atoms* required as preconditions by these actions. The heuristic $h_X(s)$ is then defined as

$$h_X(s) = Cost_X(\pi(G,s)) \quad (11)$$

where $Cost_X(\pi)$ is the sum of the action costs for the actions in π that do not affect X plus the estimated cost of the 'local plan' [4] that generates all the X-atoms in π, expressed as

$$Cost_X(\pi) = Cost(\pi \cap \bar{X}) + Cost_{TSP}(\pi \cap X) \quad (12)$$

where

$$\pi(p;s) = \begin{cases} \{\} & \text{if } p \in s \\ \{p\} & \text{if } p \in X \\ \pi(a_p;s) & \text{otherwise} \end{cases}$$

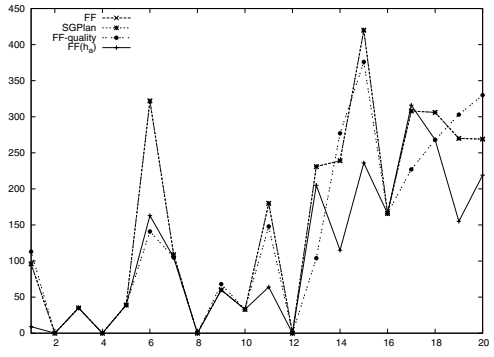
$$a_p = \mathrm{argmin}_{a \in O(p)} Cost_X(\pi(a;s))$$

$$\pi(a;s) = \{a\} \bigcup \{\cup_{q \in Pre(a)} \pi(q;s)\}$$

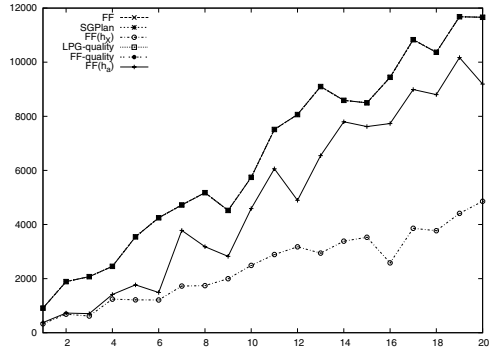
and $Cost_{TSP}(R)$ is the cost of the **best path** spanning the set of atoms R, starting from the value of X in s, in a directed graph whose nodes stand for the different values x of X, and whose edges (x, x') have costs that encode approximations of the cost of achieving x' from x' in s (see [13] for details).

We have implemented the planner FF(h_X) in which h_X, rather than h_a, is used to derive the relaxed plan, with the X variables being automatically chosen as the root variables of the causal graph [3, 9]. This planner produces plans of much lower cost than any other planner tested in the soft goals version of the *Simplified Rovers* domain (Fig. 3b), and plans of much lower cost than any other planner except LPG in the hard goals version (Fig. 3c), where LPG produces plans of only slightly worse quality.

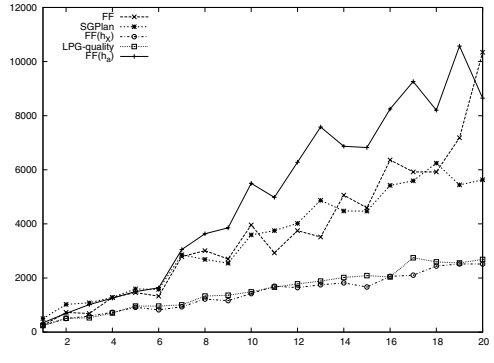
[9] In our planner we have implemented the 2-opt algorithm discussed in [14].



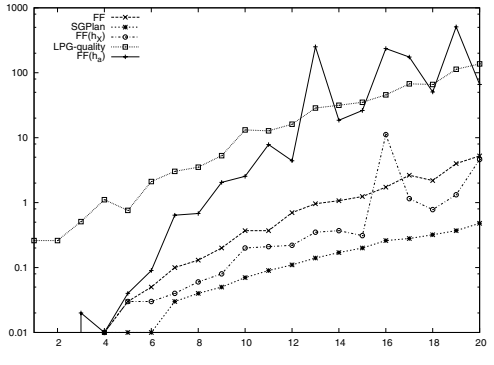
(a) Plan costs - *Colored Blocksworld* domain



(b) Plan costs - soft goals version of *S. Rovers* domain



(c) Plan costs - hard goals version of *S. Rovers* domain



(d) Planning times - hard goals version of *S. Rovers* domain

Figure 3.

5 DISCUSSION

We have shown that relaxed plans and therefore helpful actions can be computed without the use of a relaxed planning graph, meaning that other heuristics can be used in conjunction with FF's powerful EHC search. Our method of relaxed plan extraction using the additive heuristic is *cost-sensitive* and does not impose a large overhead over that of FF. Furthermore, a simple planner that combines the relaxed plan extracted in this way with the EHC search algorithm compares favourably to the state of the art in planning with action costs. Two other variations of the additive heuristic were also presented: the set-additive heuristic in which the relaxed plans are computed recursively, and the TSP heuristic, that takes delete-information into account. In both cases, labels are propagated rather than numbers in the equation characterizing the additive heuristic. Used together with EHC search, the TSP heuristic produces plans of much lower cost than any other planner tested in navigation problems where finding good paths going through a set of locations is critical. Our implementation of the TSP heuristic, however, is preliminary, and is suited only for problems where these locations correspond to the values of a single root variable in the causal graph.

ACKNOWLEDGEMENTS

We thank the reviewers for useful comments and J. Hoffmann for making the sources of Metric-FF available. H. Geffner is partially supported by grant TIN2006-15387-C03-03 from MEC/Spain.

REFERENCES

[1] A. Blum and M. Furst, 'Fast planning through planning graph analysis', in *Proc. IJCAI-95*, pp. 1636–1642, (1995).
[2] B. Bonet and H. Geffner, 'Planning as heuristic search', *Artificial Intelligence*, **129**(1–2), 5–33, (2001).
[3] R. Brafman and C. Domshlak, 'Structure and complexity of planning with unary operators', *JAIR*, **18**, 315–349, (2003).
[4] R. Brafman and C. Domshlak, 'Factored planning: How, when, and when not', in *Proc. AAAI-06*, (2006).
[5] Y. Chen, B. W. Wah, and C. Hsu, 'Temporal planning using subgoal partitioning and resolution in SGPlan', *JAIR*, **26**, 323–369, (2006).
[6] M. Do and S. Kambhampati, 'Sapa: A domain-independent heuristic metric temporal planner', in *Proc. ECP 2001*, pp. 82–91, (2001).
[7] R. Fuentetaja, D. Borrajo, and C. Linares, 'Improving relaxed planning graph heuristics for metric optimization', in *Proc. 2006 AAAI Workshop on Heuristic Search*, (2006).
[8] A. Gerevini, A. Saetti, and Ivan Serina, 'Planning through stochastic local search and temporal action graphs in LPG', *JAIR*, **20**, 239–290, (2003).
[9] M. Helmert, 'A planning heuristic based on causal graph analysis', in *Proc. ICAPS-04*, pp. 161–170, (2004).
[10] J. Hoffmann, 'The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables', *JAIR*, **20**, 291–341, (2003).
[11] J. Hoffmann and B. Nebel, 'The FF planning system: Fast plan generation through heuristic search', *JAIR*, **14**, 253–302, (2001).
[12] E. Keyder and H. Geffner, 'Heuristics for planning with action costs', in *Proc. Spanish AI Conference (CAEPIA 2007)*, volume 4788 of *Lecture Notes in Computer Science*, pp. 140–149. Springer, (2007).
[13] E. Keyder and H. Geffner, 'Set-additive and TSP heuristics for planning with action costs and soft goals', in *ICAPS-07 Workshop on Heuristics for Domain Independent Planning*, (2007).
[14] S. Lin and B. W. Kernighan, 'An effective heuristic algorithm for the TSP', *Operations Research*, **21**, 498–516, (1973).
[15] Derek Long and Maria Fox, 'Automatic synthesis and use of generic types in planning.', in *Proc. AIPS-2000*, pp. 196–205, (2000).
[16] O. Sapena and E. Onaindia, 'Handling numeric criteria in relaxed planning graphs', in *Advances in Artificial Intelligence: Proc. IBERAMIA 2004, LNAI 3315*, pp. 114–123. Springer, (2004).
[17] D. E. Smith, 'Choosing objectives in over-subscription planning', in *Proc. ICAPS-04*, pp. 393–401, (2004).