A New Approach to Heuristic Estimations for Cost-Based Planning

Raquel Fuentetaja and **Daniel Borrajo** and **Carlos Linares**
Departamento de Informática, Universidad Carlos III de Madrid
Ada. de la Universidad, 30. 28911 Leganés(Madrid). Spain
rfuentet@inf.uc3m.es, dborrajo@ia.uc3m.es, clinares@inf.uc3m.es

Abstract

Solving relaxed problems is a commonly used technique in heuristic search to derive heuristic estimates. In heuristic planning, this is usually done by expanding a planning (reachability) graph on the current search state where the delete lists of operators are removed from their definition. Usually, this technique is used to obtain plan length estimates. However, in cost-based planning the goal is to find plans minimizing some criteria. This requires the redefinition of the heuristic estimation to account for operators costs. This paper introduces a new approach to compute cost-based heuristics using planning graphs in order to overcome some problems of the existing heuristics, together with a common way of characterizing heuristics based on planning graphs. We explore the heuristics behaviour in combination to two search algorithms. Results show that in some domains the new heuristics are adequate to obtain good quality plans without imposing significant overheads in running time.

Introduction

One of the most successful techniques to guide search is using heuristic functions obtained from simplified (or relaxed) versions of problems. The most common relaxation used in STRIPS planning to guide heuristic planners is to ignore the delete effects of the domain operators. This type of relaxation was first proposed by Mcdermott (McDertmott 1996) and it has been used in many planners as HSP (Bonet & Geffner 2001), FF (Hoffmann 2003), or SAPA (Do & Kambhampati 2003). A standard technique to compute this relaxation consists of generating a relaxed graph-plan following GRAPHPLAN (Blum & Furst 1995).

Nowadays, some planners as LPG-td (Gerevini, Saetti, & Serina 2004), SAPA (Do & Kambhampati 2003), SimPlanner (Sapena & Onaindía 2004), or Metric-FF (Hoffmann 2003) deal with planning domain representations more expressive than STRIPS, like defining temporal actions or usage of numerical resources. One of these extensions consists of incorporating a quality metric in the planning problems for measuring how good a plan is. The corresponding planning model is called cost-based planning.

As this planning model becomes more expressive than the classical one (STRIPS), the heuristics and methods to compute them should account for that extra expressiveness. Nowadays, cost-based planning is an area of growing interest since probabilistic, temporal and over-subscription planning problems can be solved by transforming them into cost-based planning problems.

In this paper, we focus on the study of the behaviour of different heuristics for cost-based planning and propose a new approach to compute cost-based heuristics, characterizing all heuristics based on relaxed planning graph by defining a generalized algorithm as a common framework. We also study the impact of the different heuristics when using them in combination with different search algorithms.

Planning model and heuristics for cost-based planning

We consider a cost-based planning problem as a tuple (P, A, I, G, M), where P is the set of atoms, A is a set of grounded actions, $I \in P$ and $G \in P$ are the initial state and the set of goals and M is the metric criteria. As in STRIPS planning, each action $a \in A$ is represented as three lists: $pre(a) \subseteq P$, $add(a) \subseteq P$ and $del(a) \subseteq P$ (preconditions, adds and deletes). Each action also has a cost, $cost(a)$, that depends on the metric criteria M (in STRIPS planning this cost is uniform and equal to 1). A plan π is an ordered set of grounded actions a_i, whose cost is:

$$cost(\pi) = \sum_{\forall a_i \in \pi} cost(a_i)$$

The optimal plan for solving a cost-based planning problem is a plan π^* with the minimum cost. In this work, we are not interested on finding the optimal plan but a good plan. Instead, we focus on obtaining a good trade-off between solution cost and time to solve.

Most heuristics for cost-based planning are based on relaxed planning graphs (RPGs) or can be implemented using them. A RPG can be represented as a sequence (P_0, A_0, ...,P_{i-1}, A_{i-1}, P_i) of proposition sets (P_j) and action sets (A_j). These sets are built incrementally starting with $P_0 = I$ (initial state) as the initial layer. Then, iteratively, actions that can be applied in a given layer are inserted as A_i, and the add effects of those actions are inserted in the next proposition layer P_{i+1} together with all propositions in P_i. A general algorithm for building RPGs is shown in figure 1.

```
COMPUTE_RPG_HEURISTIC(G,I,A)
i = 0; P_0 = I;
while not end_condition() do
    A_i = compute_next_action_level()
    P_{i+1} = compute_next_proposition_level()
    if fail_condition() then return ∞
    i = i + 1
return extract_heuristic()
```

Figure 1: General algorithm for computing heuristics using relaxed planning graphs.

The computation of all heuristics we describe now follow this algorithm. They differ on how they carry out the computation of the next action and proposition levels, check the end and fail conditions and extract a heuristic value from the expanded planning graph.

METRIC-FF heuristic

METRIC-FF (Hoffmann 2003) instantiates the previous algorithm as follows:

- `end_condition()` is true at level i when all top-level goals are included in this level: $G \subseteq P_i$
- `compute_next_action_level()` returns the set of all applicable actions in the current layer i : $A_i = \{a \in A \mid prec(a) \subseteq P_i\}$
- `compute_next_proposition_level()` returns the set of literals achieved so far: $P_i \cup add(a), \forall a \in A_i$
- `fail_condition()` returns true when $P_{i+1} = P_i$, i.e. when a fixpoint is reached without finding the top-level goals
- `extract_heuristic()` extracts a sequential relaxed plan from the RPG without backtracking, and then returns the heuristic of METRIC-FF, h_{mff}, defined as the sum of the costs of all actions in that plan. Though h_{mff} is based on the costs of actions in the plan, the computation of the relaxed plan itself ignores the cost information. The admissibility of h_{mff} is not guaranteed because the extracted relaxed plan is not always optimal.

HSP extended heuristics

HSP (Bonet & Geffner 2001) heuristics can be extended to deal with cost-based planning by including the cost of each action, $cost(a)$, as in (Keyder & Geffner 2007). In that work, the authors propose a heuristic based on the additive heuristic. Instead, we use the same paradigm defined in the general procedure of Figure 1. The cost of achieving an atom p from a state s can be defined as:

$$g_s(p) = \begin{cases} 0 & \text{if } p \in s \\ \min\limits_{a \in A(p)} [cost(a) + g_s(prec(a))] & \text{otherwise} \end{cases} \quad (1)$$

where $A(p)$ stands for the actions that add p, and $g_s(prec(a))$ stands for the estimated cost of achieving the preconditions of action a from s.

The cost $g_s(C)$ of the set of atoms C is defined in terms of the cost of the atoms in the set. It can be defined as the *sum* of the costs of the individual atoms in C (h_{add}), or as the maximum cost of the atoms (h_{max}). The heuristic evaluation of a state s is $g_s(G)$, where G are the goals.

These heuristics can be implemented with RPGs in the way proposed by (Do & Kambhampati 2003), though they do not necessarily need a RPG for its implementation. Using a RPG implies instantiating the generic algorithm in Figure 1 in the same way as METRIC-FF does but:

- `compute_next_proposition_level()` also computes a propagation of costs such that each proposition p in a level has an associated cost equal to the current update of $h_{add}(p)$ (or h_{max}) in that level. These costs decrease monotonically with layers.
- `end_condition()` since the costs decrease monotonically with layers, when $G \subseteq P_i$, costs of propositions can usually be improved if the RPG is extended. The RPG can be extended by building an additional layer (1-lookahead), k layers (k-lookahead), or until a fixpoint (∞-lookahead) is reached in which both propositions and costs are stable. The formal definitions of $h_{add}(p)$ and h_{max} correspond to an ∞-lookahead
- `extract_heuristic()` does not extract a solution from the RPG, but returns $h_{add}(G)$ (or $h_{max}(G)$) of the last layer generated.

SIMPLANNER heuristic

The main difference between the heuristic employed in SIMPLANNER (Sapena & Onaindía 2004), h_{sim}, and h_{mff} is that in the former, levels of the graph do not represent time steps, but costs according to the problem metric. In h_{mff}, the estimation for a cost-based planning problem is computed from a RPG built to minimize parallel plan length. But, if the goal is to minimize a given cost function, it seems better to expand the RPG based on costs for achieving propositions/actions and not on the number of operators to achieve it. While h_{sim} minimizes the cost for achieving literals, a method for minimizing the costs of actions is introduced in the next section.

In the case of SIMPLANNER:

- `compute_next_action_level()` returns the set of all applicable actions whose preconditions have costs less than or equal to a value $cost\_limit_i$. So,

$$A_i = \{\, a \in A \mid prec(a) \subseteq P_i \wedge \forall p \in prec(a), cost(p) \leq cost\_limit_i \}$$

The $cost\_limit_i$ of layer i is the minimum cost of all propositions achieved up to that layer not yet in the RPG. A global list, L, with the current minimum cost for every achieved proposition is maintained to compute $cost\_limit_i$. Initially, L has the propositions in the initial state (whose cost is 0). Propositions not appearing in L have cost ∞.

The execution of actions modifies L, instead of the next proposition layer: everytime an action a achieves a proposition p not yet included in the RPG its cost is updated in L as:

$$cost(p) = min[cost(p), cost(a) + \sum_{r \in prec(a)} cost(r)] \quad (2)$$

After the execution of all the actions in A_i, $cost\_limit_{i+1}$ is computed as the cost of the proposition with minimum cost in L. Propositions with cost equal to $cost\_limit_{i+1}$ are then deleted from L, and:

- `compute_next_proposition_level()` returns the union of the set of propositions in L whose cost is $cost\_limit_{i+1}$ and the propositions in the previous level P_i.
- `fail_condition()` returns true when L is empty (i.e. when a fixpoint is reached).

The `end_condition()` function is the same as in h_{mff}. The `extract_heuristic()` function is also similar except for minor implementation details.

Figure 2 shows an example of the cost limit computed for each level in the RPG of h_{sim}. There are five actions: a_1(pre:$\{p\}$, add:$\{q, r\}$, cost:15), a_2(pre:$\{p\}$, add:$\{s\}$, cost:20), a_3(pre:$\{q, r, s\}$, add:$\{t\}$, cost:10), a_4(pre:$\{t\}$, add:$\{k\}$, cost:2), and a_5(pre:$\{p\}$, add:$\{k\}$, cost:50). In the initial state we have p, and the goal is k. The actions in A_0 add their effects to the list L; a_1 adds q and r with cost 15, a_2 adds s with cost 20, while a_5 adds k with cost 50. So, the next proposition level is composed of p and least costly to achieve propositions, q and r, delaying the achievement of s and k to later. The execution of the action a_3 includes t in L but it does not include t in P_3. When the goal k appears in P_3, the solution plan is extracted as indicated by the arrows. The cost of this solution (h_{sim}) is 50. The heuristics h_{mff}, h_{add} and h_{max} can be easily computed for this example, yielding the values 50, 50, and 32, respectively.

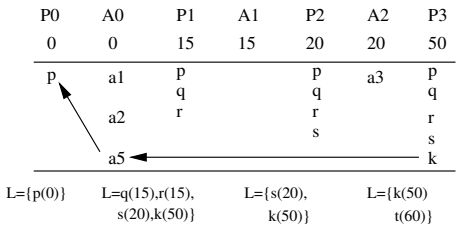

Figure 2: Example of the RPG for h_{sim}.

Theorem 1. If a proposition p appears for the first time in proposition level i, the cost limit of the next action level, $cost\_limit_i$, is $h_{add}(p)$. The heuristics h_{sim} and h_{add} assume subgoal independence. They are different in that h_{sim} is computed from an extracted sequential relaxed plan as described in h_{mff}, while an implementation of h_{add} over RPGs does not extract the plan, but returns the sum of the propagated cost for each goal (i.e. the cost that the goal has in the last proposition layer).

Level-based heuristics

In our approach, we build the RPG based on delaying actions instead of propositions as in h_{sim}, using an idea similar to the Dijkstra algorithm. First, we compute the applicable actions on the initial state, App_0 (their preconditions are in P_0). Then, we generate A_0 to contain only those actions in App_0 whose cost is minimum; that is, $A_0 = \{a \in App_0 \mid a \in \arg cost\_limit_0\}$, where $cost\_limit_0 = \min_{a \in App_0} cost(a)$. In general, $cost\_limit_i$ represents the cost of the actions in A_i plus the cost of their preconditions. The rest of actions in App_0 are added to a delayed set of actions $D = App_0 - A_0$. The next set of propositions P_1 is computed as in METRIC-FF by adding to P_0 the propositions in the add lists of actions in A_0. We then compute the set of actions that can be applied, App_1, by the union of those new actions that can be applied from the propositions in P_1 and the delayed actions, D. Again, the next action layer, A_1, is computed by selecting the less costly actions from that set: $A_1 = \{a \in App_1 \mid a \in \arg cost\_limit_1\}$, and a new delayed set D is computed. The process continues until all goals are true in a given proposition layer. With this solution we do not assume atoms independence as in the case of h_{sim} and h_{add}.

- `compute_next_action_level()` returns the set

$$A_i = \{a \in App_i \mid a \in \arg cost\_limit_i\}$$

We provide two ways of computing $cost\_limit_i$ in each action level, giving rise to two different heuristics h_{level1} and h_{level2}:

- h_{level1}: $cost\_limit_i = min_a(cost(a) + cost\_limit_{i-1})$, where a are the applicable actions not yet in the RPG. Here we assume that the cost of the preconditions of a is the $cost\_limit$ of the previous level.
- h_{level2}: $cost\_limit_i = min_a(cost(a) + cost\_limit_k)$, where a are the applicable actions not yet in the RPG, and $k + 1$ is the first proposition level in which all the preconditions of a appear. Here we assume that the cost of the preconditions of a is given by the first level in which all of them appear.
- `fail_condition()` returns true when $P_{i+1} = P_i$ and there are no applicable actions not yet in the RPG.

The `end_condition()`, `compute_next_proposition_level()` and `extract_heuristic()` functions are as in h_{mff}.

The Figure 3 shows how the cost limits corresponding to h_{level1} and h_{level2} are computed for the example of the previous section. The $cost\_limit_0$ (15) is the cost of the least costly action (a_1). A_0 only contains a_1. When a_1 is applied it adds q and r to P_1. Then, the $cost\_limit_1$ is computed:

- For h_{level1}, as the cost limit of the previous layer (15) plus the cost of the least costly action, 20 (given by a_2).
- For h_{level2}, as the cost of a_2 (20) because it is the least costly action and all its preconditions are true initially.

When the goal k appears in P_4, the solution plan is extracted as indicated by the arrows. The cost of this solution is 47.

For this example h_{level1} and h_{level2} are the same and they are the only heuristics that predict exactly the cost of the optimal solution.

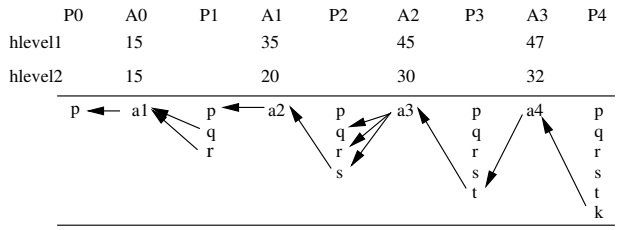

Figure 3: Examples of the RPG for h_{level1} y h_{level2}.

Theorem 2. In the case of h_{level2}, if a proposition p appears for the first time in proposition level $i+1$, the cost limit of the previous action level, $cost\_limit_i$, is $h_{max}(p)$.

In spite of this theorem, h_{level2} is not equal to h_{max} because h_{level2} is computed from an extracted sequential relaxed plan as described in h_{mff}, while $h_{max}(G)$ returns the cost of the last layer expanded.

Theorem 3. In the case of h_{level1}, if a proposition p appears for the first time in proposition level $i+1$, the cost limit of the previous action level is between $h_{max}(p)$ and $h_{add}(p)$: $h_{max}(p) \leq cost\_limit_i \leq h_{add}(p)$.

When optimizing plan length, the algorithm behaves as METRIC-FF. Table 1 summarizes all the described heuristics in accordance with the general algorithm depicted Figure 1. The column "Levels of the RPG" synthesizes the functions `compute_next_action_level()` and `compute_next_proposition_level()`. It can be: classical, classical with a process of cost propagation or cost-oriented. The `fail_condition()` is not included because it is similar for all cases. The only *admissible* heuristic is h_{max}.

Table 1: Differences between heuristics.

h	`end_condition()`	Levels of the RPG	`extract_heuristic()`
h_{add}	0-lookahead	classical	costs of goals
h_{max}	∞-lookahead	(costs prop.)	(last RPG level)
h_{mff}	$G \subseteq P_t$	classical	relaxed plan
h_{sim}	$G \subseteq P_t$	cost-oriented	relaxed plan
h_{level1}	$G \subseteq P_t$	cost-oriented	relaxed plan
h_{level2}	$G \subseteq P_t$	cost-oriented	relaxed plan

Search algorithms

In order to compare the behaviour of the previous heuristics, we will use two search algorithms on top of METRIC-FF: weighted-A* and Cost-EHC (CEHC). This section introduces CEHC, an adaptation of Enforced Hill Climbing (EHC) we propose, suitable to deal with cost-based planning problems.

Enforced Hill Climbing (EHC) is the algorithm used by METRIC-FF when the metric is plan length. For each selected node n, EHC uses a forward breadth-first search until it finds a successor node in the subtree that returns a better heuristic estimate than n. This will become the next selected node, and search continues. If no solution is found, it switches to weighted-A*. If a metric criteria different of plan length is given, METRIC-FF uses a standard weighted-A*. However, weighted-A* is usually too expensive both in time and memory, and as shown in the experimental section, only a few problems can be solved. For this reason, we use EHC for any metric criteria, but breaking ties in favour of shorter solutions. So, we compute for each state two different heuristics using the same relaxed plan: the sum of the cost of each action and the plan length. We call this algorithm Cost-EHC (CEHC).

EHC reduces the branching factor by prunning actions that are not *helpful*. The *helpful actions* are computed during the extraction of a relaxed plan as applicable actions in the initial state that add a relevant subgoal for the relaxed plan: if the sets of relevant subgoals in each level are $G_0, G_1, \ldots, G_{final\_layer}$, the helpful actions achieve propositions in G_1. To compute the helpful actions we follow the same philosophy as METRIC-FF, but ordering them by increasing order of costs. Besides, we take into account that actions delayed in the first level (actions whose preconditions are true, but are not executed due to the $cost\_limit$ condition) of the graph can achieve relevant subgoals in a set different of G_1. Hence, we re-define the set of helpful actions as:

$$H(s) = \{a \in A \mid prec(a) \subseteq I, \text{add}(a) \cap G_i \neq \emptyset, i \geq 1\}$$

Thus, an action is considered helpful when its preconditions are true in the initial state and it achieves propositions in any set of relevant subgoals different of G_0.

A similar idea can be applied to h_{sim}. However, the idea of helpful actions is not applicable when the process of extracting the heuristic value does not obtain a relaxed plan from the RPG (h_{add} and h_{max}).

Experimental results

In this section, we present a summary of the results obtained from different experiments, using an implementation of all the aforementioned heuristics in METRIC-FF. The domains and problems used herein are some numeric domains from the 3rd International Planning Competition (IPC3).[1] In later IPCs almost all domains include durative actions and the study of its management in a numerical setting is outside the scope of this paper. We chose the domains from the IPC3 where more problems were solved: the *Zenotravel*, *Driverlog*, and *Depots* domains. Among other domains that we did not use are: the *Satellite* domain, because there are many *dead-ends* and the planner fails using all heuristics in almost all problems; and the *Rovers* domain, where the metric is restricted to the number of recharges and only one action has cost greater than one. For uniformity, we modified the problems of the same domain to have all the same metric expression. Thus, we used the following as metric expressions to minimize: *fuel_used* in *Zenotravel*, *number-of-operators* + *driven* + *walked* in *Driverlog* and *fuel_cost* in *Depots*.

[1] http://planning.cis.strath.ac.uk/competition/

We ran the experiments on a Linux machine running at 2 GHz. The maximum memory allowed for the planners was 500Mb. The maximum time allowed for the planners to find a solution was set to 300 seconds (this is the time bound usually used in IPCs).

Results on solvability

Firstly, Table 2 shows the number of problems solved by all configurations, i.e. all heuristics with A* and the CEHC algorithm. The number aside each domain name refers to the total number of test problems. As expected, the A* algorithm fails to find a solution within the time bound on a large number of problems. The number of problems solved by CEHC in all cases (except for h_{max}) is very similar LPG-td(quality) (Gerevini, Saetti, & Serina 2004) (in the second column), which is a competitive cost-based planner. In the Zenotravel domain CEHC with h_{level1} solves all problems.

Table 2: Problems solved by all configurations.

Domain	LPG	A* / CEHC					
		mff	$level1$	$level2$	sim	add	max
Zeno(20)	20	10/18	8/20	7/17	10/16	12/15	4/10
Dlog(20)	15	3/16	4/15	4/15	4/15	9/15	1/12
Depot(22)	21	6/21	8/21	8/21	4/21	14/19	2/7
Total(%)	90	30/89	32/90	31/85	29/84	56/77	11/47

Heuristics quality

Secondly, we wanted to estimate how far all the previous heuristics are from the optimal values. In order to compute this, we would need to compute the cost of the optimal solution from each node in the search tree and compare it against each heuristic estimation. Instead, we propose here a simplified way of computing an estimation of the distance of each heuristic value to an estimated optimal value, as depicted in Figure 4. We selected the Zenotravel, and for each node in the search tree of 50 randomly generated problems, we compared the cost of the best solution found from that node minus the g value of that state ($h_{best\_solution}$), in the x-axis, with the h estimate of this state, in the y-axis. Estimations above the main diagonal are non-admissible. Estimations below the diagonal could also be non-admissible, because we do not compare against the optimal solution cost, but at least they are under-estimates of the best cost found so far. For these problems the most accurate heuristic estimations were achieved with h_{level1} followed by h_{sim}, in contrast with h_{mff} that produces the least accurate estimations. We use the Mean Absolute Error (MAE) $MAE = \frac{1}{N}\sum|h_{best\_solution}(s) - h_i(s)|$, where N is the total number of states. The MAEs for all heuristics are: 84.5 (h_{level1}), 128.2 (h_{sim}), 269.8 (h_{level2}), 342 (h_{add}), 562.1 (h_{max}) and 1875,5 (h_{mff}). This type of information can be very helpful, for instance, to provide an idea of what w to use in a weighted-A^* search. Taking into account that the real value of heuristics for planning problems is how well they select "optimal" actions from the rest and not how precise they are, it would be interesting to determine how related are the absolute accuracy of the heuristics and their goodness on ranking correctly the actions.

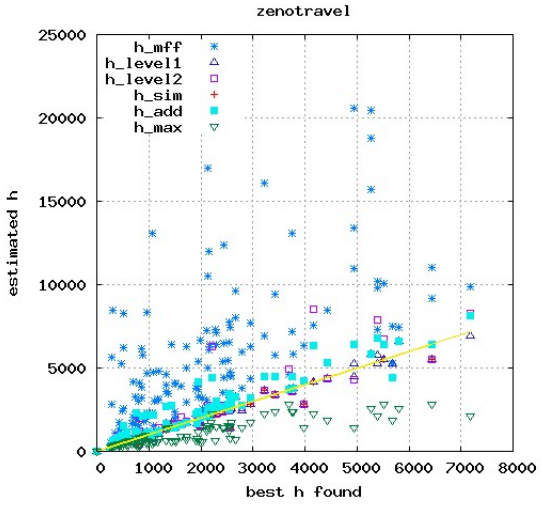

Figure 4: Differences among heuristic values in relation to the best cost of some solutions paths found with A* in the Zenotravel domain.

Experiments with A*

Thirdly, we wanted to compare the quality of the solutions found using the standard weighted-A* for all heuristics. We compare only the quality of the problems solved by all configurations, and this number was quite small for all domains: 4, 2, and 3 for Zenotravel, Driverlog and Depots, respectively. For this reason we only provide some results for the Zenotravel domain without taking into account h_{max} (thus we have 7 solved problems). For these problems, all heuristics provide a positive accumulated **cost gain** with respect to h_{mff} (we take the solution of METRIC-FF as a reference for the comparisons). Results are shown in Table 3. Columns are from left to right: domain (along with the number of problems solved by all configurations), the heuristic function tested, the percentage of cost gain with respect to using the standard METRIC-FF heuristic, the percentage of problems that each version solved with equal, better and worse quality than METRIC-FF respectively, and finally the comparison in terms of time and number of nodes with respect to METRIC-FF where $\times n$ means that the time (or number of nodes) of h_{mff} is multiplied by n.

Table 3: Comparison against h_{mff} using A* in the Zenotravel domain.

Domain	h	cost gain (%)	equal (%)	better (%)	worse (%)	CPU time	#nodes
Zeno(7)	$level1$	13.5	42.8	57.1	0	$\times$1.6	$\times$1.2
	$level2$	13.5	42.8	57.1	0	$\times$4.1	$\times$1.6
	sim	4.0	42.8	42.8	14.2	$\times$7.8	$\times$2.0
	add	3.0	42.8	28.5	28.5	$\times$2.6	$\times$1.2

We can see that the two new heuristics provide a reasonable trade-off between the quality of solutions (they obtained the best results in all problems and the relation $time/nodes$ to compute it (only $1.6/1.2$ the behaviour of METRIC-FF).

In the Zenotravel domain subgoals are not independent. To board several people in a location the plane should be in the same location, but it is not necessary to add the cost of flying the plane to that location for each person. Another important thing is how the action costs are distributed. In this domain the costs of the operators *zoom* and *fly* are considerably greater than zero, which is the cost of the operator *board*. This fact further increases the error of assuming subgoal independence. This explains why the heuristics h_{level1} and h_{level2} obtain better results than h_{sim} and h_{add}.

Experiments using CEHC

The next step is to compare with h_{mff} using the CEHC algorithm. A summary is shown in Table 4. We do not include the results for h_{max} in the Zenotravel and Depots domains because this heuristic restricts a lot the number of problems solved. In all the tested domains the accumulated cost is better using any heuristic than using h_{mff}.

Table 4: Comparison against h_{mff} using CEHC.

Domain	h	cost gain (%)	equal (%)	better (%)	worse (%)	CPU time	#nodes
Zeno (15)	$level1$	42.6	6.6	80	13.3	×1.4	×1.3
	$level2$	45.3	6.6	86.6	6.6	×8.3	×1.7
	sim	47.3	6.6	86.6	6.6	×6.9	×1.2
	add	35.6	6.6	93.3	0	×56.7	×5.8
Dlog (12)	$level1$	1.6	16.6	50	33.3	×1.4	×0.6
	$level2$	7.5	8.3	75	16.6	×8.8	×1.8
	sim	10.9	8.3	58,3	33.3	×1.9	×0.7
	add	19.5	0.0	83.3	16.6	×4.9	×0.77
	max	1.5	0.0	66.6	33.3	×44.2	×10.9
Depots (19)	$level1$	18.9	15.8	73.7	10.5	×1.1	×1.9
	$level2$	18.9	15.8	73.7	10.5	×1.1	×1.9
	sim	30.2	21.0	63.1	15.8	×0.8	×0.4
	add	29.0	15.8	68.4	15.8	×3.5	×0.9

For the Zenotravel domain, the best result in cost gain is obtained using h_{sim}, though h_{level1} and h_{level2} perform very similarly. However, the CPU time with h_{level1} is only 1.4 times the time with h_{mff}, while with h_{sim} the time is multiplied by 6.9. In terms of number of nodes, these heuristics perform very similar to h_{mff}.

In the Driverlog domain, the best results in cost gain are obtained with h_{add} and h_{sim}. Both heuristics increase a little the CPU time, though they decrease the number of nodes. Therefore, the increase in the CPU time is due to the heuristic computation (not to the search).

Finally, in the Depots domain, h_{sim} is the heuristic with best cost gain. Besides, it reduces the time and the number of expanded nodes of METRIC-FF. In this domain h_{add} also performs well in terms of cost gain, though it increases the CPU time 3.5 times.

CEHC is a search algorithm strongly influenced by the order in which actions are selected, and therefore it is less sensitive than A* to changes in the heuristic evaluation. This makes it difficult to explain the obtained results from the point of view of domain-problem characteristics (though they show that METRIC-FF can be easily improved in terms of quality).

Conclusions

In this paper we characterize the heuristics for forward search in cost-based planning following a generalized algorithm. Using this characterization we introduce two new heuristics that try to provide a reasonable tradeoff between solution cost and time to solve and we establish their relation to the previous heuristics. We report experiments using two search algorithms (A* and CEHC). CEHC is a new adaptation of EHC for dealing with costs. The notion of *helpful actions*, used to prune the search, is also adapted for the case of our heuristics. We also propose a way of characterizing heuristics quality by comparing heuristic values against possibly good solution costs.

Results show that the new heuristics improve over METRIC-FF in terms of quality, while not incurring in a significant overhead in the time to solve. We would like to compare also with LPG-td(quality), using as the search algorithm a combination of CEHC and A*. Next, we would like to define and compare against more heuristics and search algorithms in order to understand their behaviour with respect to domain characteristics.

Acknowledgements

This work has been partially supported by the Spanish MEC project TIN2005-08945-C06-05.

References

Blum, A. L., and Furst, M. L. 1995. Fast planning through planning graph analysis. *IJCAI-95*, volume 2, 1636–1642.

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

Do, M. B., and Kambhampati, S. 2003. Sapa: A scalable multi-objective heuristic metric temporal planner. *JAIR* 20:155–194.

Gerevini, A.; Saetti, A.; and Serina, I. 2004. Planning with numerical expressions in LPG. *ECAI-04*, 667–671.

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. 2007. Heuristics for planning with action costs. In *Proceedings of the 12th Conference of the Spanish Association for Artificial Intelligence (CAEPIA)*.

McDermott, D. 1996. A heuristic estimator for means-ends analysis in planning. In *AIPS-96*, 142–149.

Sapena, O., and Onaindía, E. 2004. Handling numeric criteria in relaxed planning graphs. In *Advances in Artificial Intelligence. IBERAMIA, LNAI 3315*, 114–123.