The FF(h_a) Planner for Planning with Action Costs

Emil Keyder
Universitat Pompeu Fabra
Passeig de Circumvalació 8
08003 Barcelona Spain
emil.keyder@upf.edu

Héctor Geffner
ICREA & Universitat Pompeu Fabra
Passeig de Circumvalació 8
08003 Barcelona Spain
hector.geffner@upf.edu

Abstract

FF(h_a) is the FF planner but with a different heuristic h_a that is sensitive to action costs. More precisely, the relaxed plans in FF(h_a) are computed in a simple manner from the additive heuristic h_a rather than from the relaxed planning graph as in FF. The result is a planner that inherits the properties of FF, in particular, its performance, speed, and scalability, but that takes cost information into account.

Introduction

FF(h_a) is a heuristic search planner that combines elements of FF (Hoffmann & Nebel 2001) and HSP (Bonet & Geffner 2001). The heuristics used are variations of HSP's additive heuristic h_a, while the basic search algorithm, enforced hill climbing (EHC), is that of FF. The result is a planner that can generate plans as quickly as FF but that is cost-sensitive; that is, it is able to take into account cost metrics other than the number of actions in a plan during the search. Results on several domains and a more detailed description of the planner are available in (Keyder & Geffner 2008); here we give only a short overview.

In order to simplify the definition of the heuristics, we introduce a 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 estimate $h(s)$ of the cost from state s to the goal is then written as $h(G; s)$, the cost of achieving the 'dummy' atom G from s.

Additive Heuristic h_a

The additive heuristic introduced in (Bonet & Geffner 2001) is a polynomial approximation of the optimal delete-free heuristic h^+ where subgoals are assumed to be *independent*. This assumption is normally false but results in an heuristic function $h_a(s)$ that is easy to compute where

$$h_a(s) \;=\; h(G; s) \tag{1}$$

and $h(p; s)$ provides an estimate of the cost of achieving p from s given by the equations

$$h(p; s) \;=\; \begin{cases} 0 & \text{if } p \in s \\ h(a_p; s) & \text{otherwise} \end{cases} \tag{2}$$

with

$$a_p \;=\; \text{argmin}_{a \in O(p)} h(a; s) \tag{3}$$

and

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

where $O(p)$ stands for the set of actions in the problem that add p.

The additive heuristic, as its name implies, makes the assumption that the cost of achieving a set of atoms is equal to the *sum* of the costs of achieving each of the atoms separately. When this assumption is true, either because action preconditions and subgoals can be achieved with *no side effects*, or because the goal and action preconditions contain *one atom at most*, h_a is equal to h^+, and the additive heuristic is optimal for the delete relaxation. Versions of the additive heuristic appear also in (Do & Kambhampati 2001; Sapena & Onaindia 2004; Smith 2004), where the cost of joint conditions in action preconditions or goals is set to the sum of the costs of each condition in isolation.

Relaxed Plans from the Additive Heuristic

A relaxed plan $\pi_a(s)$ can be computed from the additive heuristic h_a in the state s by storing, during the computation, the supports a_p of each atom p in the problem ; namely, the actions a that add p and have minimum $h(a; s)$ value.[1] The relaxed plan $\pi_a(s)$ can be then defined in terms of these supports as:

$$\begin{aligned} \pi_a(s) \;&=\; \pi_a(G; s) \\ \pi_a(p; s) \;&=\; \begin{cases} \{\} & \text{if } p \in s \\ \{a_p\} \bigcup \pi_a(a_p) & \text{otherwise} \end{cases} \end{aligned}$$

where

$$\pi_a(a) \;=\; \cup_{q \in pre(a)} \pi_a(q; s)$$

While $\pi_a(s)$ is a *set* of actions, it is easy to show that it can be ordered as a plan that achieves the goal G from s in the delete relaxation. The heuristic estimator is then defined as

$$h^d_a(s) \;=\; \sum_{a \in \pi_a(s)} cost(a)$$

[1] Ties between supports are broken arbitrarily.

rather than as the length $|\pi_a(s)|$ of the relaxed plan. The heuristic $h^d_a(s)$ is equal to the additive heuristic $h_a(s)$ when the same action is not used to support more than one precondition in the relaxed plan $\pi_a(s)$. Otherwise, the heuristic $h^d_a(s)$ counts the cost of the actions that support multiple conditions in the relaxed plan $\pi_a(s)$ only once, while the additive heuristic $h_a(s)$ may count the cost of such actions multiple times. For this reason, when needed, we refer to the heuristic $h^d_a(s)$, as the *additive heuristic with duplicates eliminated.*

The Set-Additive Heuristic

In addition to the additive heuristic, the FF(h_a) planner implements two other heuristics for computing relaxed plans. The first such heuristic minimizes the cost of the relaxed plan for each atom *recursively* during the computation, rather than minimizing the value of the original additive heuristic during the computation and extracting the relaxed plan as a post-processing step. In the additive heuristic, the cost of the best supporter a_p of p in s, $h(a_p; s)$, is propagated to obtain the cost of p, $h(p; s)$. In contrast, in the *set-additive* heuristic, the best supporter a_p of p is itself propagated, and supports are combined by *set-union* rather than by *sum*, resulting in a function $\pi(p; s)$ that represents a *set of actions* which can be defined similarly to $h(p; s)$:

$$\pi(p;s) = \begin{cases} \{\} & \text{if } p \in s \\ \pi(a_p;s) & \text{otherwise} \end{cases} \quad (5)$$

where

$$a_p = \text{argmin}_{a \in O(p)} Cost(\pi(a;s)) \quad (6)$$

$$\pi(a;s) = \{a\} \bigcup \{\cup_{q \in Pre(a)} \pi(q;s)\} \quad (7)$$

$$Cost(\pi(a;s)) = \sum_{a' \in \pi(a;s)} cost(a') \quad (8)$$

That is, the best supporter a_p of p is propagated to compute $\pi(p; s)$, and supports for preconditions and goals are combined by set-union. The action a_p minimizing the cost of the plan made up of itself and the union of the relaxed plans for each of its preconditions is selected as the best supporter for each p. The *set-additive heuristic* $h^s_a(s)$ for a state s is then defined as

$$h^s_a(a) = Cost(\pi(G;s)) \quad (9)$$

The TSP Heuristic

The heuristic $h^s_a(s)$ associates with every atom p a relaxed plan $\pi(p; s)$ (Equations 5–9). The latter 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, …, 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, …, r_n to be picked up at locations l_1, …, l_n, the set of atoms $at(l_0)$, $at(l_1)$, …, $at(l_n)$, 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. Here we have implemented the 2-opt algorithm described in (Lin & Kernighan 1973). By comparison, the delete-relaxation approximates the cost of the problem as the cost of the best *tree* rooted at t_0 that spans all of the locations. The modification of the labels $\pi(a; s)$ 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.

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

$$h_X(s) = Cost_X(\pi(G, s)) \quad (10)$$

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' (Brafman & Domshlak 2006) that generates all the X-atoms in π, expressed as

$$Cost_X(\pi) = Cost(\pi \cap \bar{X}) + Cost_{TSP}(\pi \cap X) \quad (11)$$

where

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

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

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

and $Cost_{TSP}(P)$ is the cost of the **best path** spanning the set of atoms P, starting from the value of X in s.

Our current implementation of this TSP heuristic chooses the multi-valued variables X to be the root variables of the causal graph (Helmert 2004). This works well in certain domains in which an agent can move around a domain without restrictions and much of the cost of the problem is due to these movements. How to identify these types of domains and use this heuristic effectively when X is not the root variable is a topic of further research.

Discussion

The three heuristics discussed here are available as command-line options in the FF(h_a) planner. We have found that while the set-additive heuristic can in theory produce more accurate estimates than the more naive duplicate-eliminating additive heuristic, this effect is rarely noticeable

for current benchmark domains. h^s_a is however noticeably slower in terms of computation time, due to the addition in h_a being replaced by the set-union operation. This makes the duplicate-eliminated additive heuristic a better choice in general as more nodes can be evaluated in the same amount of time. The third heuristic h_X is useful in certain domains in which a single multi-valued variable contributes greatly to the cost of plans, but it is unclear how to detect this condition in a general way. The default heuristic used by the planner is therefore the duplicate-eliminated additive heuristic.

Planner

The main search algorithm used by FF(h_a) is EHC, with two minor changes from the version used by FF.

In FF, helpful actions are defined as $H(s) = \{a \in A | add(a) \cap G_1 \neq \emptyset\}$, where G_1 denotes the set of atoms in the first layer of the planning graph arising from the extraction of the plan $\pi_{FF}(s)$. They are defined similarly in FF(h_a), with G_1 being defined equivalently as the set of atoms p achievable in one step, i.e., $|\pi(p; s)| = 1$, such that p is a goal or a precondition of some action in the relaxed plan $\pi(G; s)$.

Second, while EHC search in FF commits to an action as soon as a node with a lower heuristic value is found, FF(h_a) generally evaluates the full set of neighboring states s', choosing among those with lower heuristic estimates than the current state the one which minimizes the expression $cost(a) + h(s')$. The exception to this is if an action is found such that the resulting state has a heuristic estimate equal to the cost of the current state minus the cost of the applied action, and the size of the relaxed plan $|\pi(s')|$ is one less than the size of the relaxed plan for the current state. If no neighboring state can be found with a lower heuristic estimate than that of the current state, EHC search at further levels continues as in FF, moving to a state with lower cost than that of the current one as soon as one is found.

FF(h_a) is implemented on top of the code for Metric-FF (Hoffmann 2003). We have used Metric-FF rather than FF as cost information is currently expressed in PDDL through numeric variables. FF(h_a), however, does not deal with numeric variables. After being used together with the plan metric to determine action costs, all numeric variables in the problem are removed from consideration.

References

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

Brafman, R., and Domshlak, C. 2006. Factored planning: How, when, and when not. In *AAAI-06*.

Do, M. B., and Kambhampati, S. 2001. Sapa: A domain-independent heuristic metric temporal planner. In *Proc. ECP 2001*, 82–91.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *Proc. ICAPS-04*, 161–170.

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

Hoffmann, J. 2003. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *J. Artif. Intell. Res. (JAIR)* 20:291–341.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In *18th European Conference on Artificial Intelligence (ECAI-08)*.

Lin, S., and Kernighan, B. W. 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research* 21:498–516.

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

Smith, D. E. 2004. Choosing objectives in over-subscription planning. In *Proc. ICAPS-04*, 393–401.