1 HSP: Heuristic Search Planner

1.1 Introduction

HSP is a planner based on the ideas of heuristic search.[1] Heuristic search algorithms perform forward search from an initial state to a goal state using an heuristic function that provides an estimate of the distance to the goal. The 8-puzzle is the standard example of heuristic search and is treated in most AI textbooks [9, 10]. The main difference between the 8-puzzle and our approach to planning is in the heuristic function. While in domains specific tasks like the 8-puzzle the heuristic function is given (e.g., as the sum of the Manhattan distances); in domain independent planning, it has to be derived from the high-level representation of the actions and goals.

1.2 Heuristic

A common way to derive an heuristic function $h(s)$ for a problem P is by relaxing P into a simpler problem P' whose optimal solution can be computed efficiently. Then, the optimal cost for solving P' can be used as an heuristic for solving P [10]. For example, if P is the 8-puzzle, P' can be obtained from P by allowing the tiles to move into *any* neighboring position. The optimal cost function of the relaxed problem is precisely the Manhattan distance heuristic.

In Strips planning, we obtain the heuristic values for a planning problem P by considering the 'relaxed' planning problem P' in which all *delete lists* are ignored. In other words, P' is like P except that delete lists are assumed empty. As a result, actions may add new atoms but not remove existing ones, and a sequence of actions solves P' when all goal atoms have been generated. As in recent planners such as SATPLAN [5] and GRAPHPLAN [1], we assume that action schemas have been replaced by all their ground instances, and we do not deal with variables.

It is not difficult to show that for any initial state s, the optimal cost $h'(s)$ to reach a goal in P' is a lower bound on the optimal cost $h^*(s)$ to reach a goal in the original problem P. We could thus set the heuristic function $h(s)$ to $h'(s)$ and hence obtain an informative and *admissible* (non-overestimating) heuristic. The problem, however, is that computing $h'(s)$ is still NP-hard.[2] We thus set for an approximation: we set the heuristic values $h(s)$ to an *estimate* of the optimal values $h'(s)$ of the relaxed problem. These estimates are computed as follows.

Starting with $s_0 = s$ and $i = 0$ we expand s_i into a (possibly) larger set of atoms s_{i+1} by combining the atoms in s_i with the atoms that can be generated by the actions whose preconditions hold in s_i. Every time we apply an action that asserts an atom p, we update a measure $g_s(p)$ that aims to estimate the difficulty (number of steps) involved in achieving p from s. For atoms $p \in s$,

[1] Blai Bonet and Héctor Geffner, Depto. de Computación, Universidad Simón Bolívar, Aptdo. 89000, Caracas 1080-A, Venezuela. Email: {bonet,hector}@usb.ve.

[2] This was first pointed out by Bernhard Nebel.

this measure is initialized to 0, while for all other atoms $g_s(p)$ is initialized to ∞. Then when an action with preconditions $C = r_1, r_2, .., r_n$ that asserts p is applied, $g_s(p)$ is updated as:

$$g_s(p) := \min \; [\; g(p)\;,\; 1 + \sum_{i=1,n} g_s(r_i)\;]$$

The expansions and updates continue until these measures do not change. It can be shown that this procedure computes the function

$$g_s(p) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s \\ i & \text{if } [\min_{C \to p} \sum_{r_i \in C} g_s(r_i)] = i - 1 \\ \infty & \text{otherwise} \end{cases} \tag{1}$$

where $C \to p$ stands for the actions that assert p and have preconditions $C = r_1, \ldots, r_n$. Then, if G stands for the goal atoms, the heuristic function $h(s)$ is defined as:

$$h(s) \stackrel{\text{def}}{=} \sum_{p \in G} g_s(p)$$

All these definitions assume, like decompositional planners, that subgoals are *independent.* The added value of the heuristic approach is that subgoals are weighted by a 'difficulty' measure that allows to regard certain decompositions as better than others. A result of this assumption, is that the heuristic function $h(s)$ is not *admissible.* On the other hand, it is often very informative and can be computed reasonably fast.

1.3 Algorithm

The heuristic function defined above allows us to deal with any Strips planning problem as a problem of heuristic search. This means we could do planning using algorithms such as A^*. The problems with A^*, however, are well known [7]. On the one hand it may take exponential space; on the other, it may proceed too slowly towards the goal. An alternative that avoids these problems is *weighted* A^* where the evaluation function is modified to $f(s) = (1 - w) \cdot g(s) + w \cdot h(s)$, where w is a real parameter, normally between 1/2 and 1 (see also [4]). An even simpler alternative is *greedy* or *hill-climbing* search, where the best node is expanded and all other nodes are discarded (ties are broken randomly). The greedy search works well in the majority of the planning problems we have considered, and in most cases yields solutions of high-quality (optimal or near optimal) very fast (in a few seconds), even in problems that are very large (such the logistic problems). Sometimes, however, it gets stuck in local minima. In such cases, it's often convenient to proceed with the search until a number of such impasses has been encountered, restarting the search if needed a given number of times. The algorithm used in the competition is a variation of this idea.

1.4 Implementation

HSP is implemented in C. Likewise, a preprocessor converts any Strips problem in PDDL into a C program that is then compiled, linked, assembled and ran. This usually means a time overhead in the order of a second or two in small planning problems but pays off in larger ones.

1.5 Results

In the competition, HSP did best in the number of problems solved. It solved 91 problems of out of a total of 165: almost 20 problems more than the other planners. On the other hand, HSP did worst in terms of time. While on average it took less than 10 seconds to solve each problem, it was considerably slower than the other planners. At the same time, while the other planners produce plans that are guaranteed to be *optimal* in terms of the number of time slices, HSP offers no such guarantees. In some cases, however, HSP generated the shortest sequential plans, and in most other cases, the generated plans were close (10%) to the shortest plans found.

1.6 Related Work

HSP is based on the planner reported in [2]. Such a planner, called ASP, uses the same heuristic function but a different search algorithm based on Korf's LRTA* [6]. It was focused on closed-loop planning and the use of a more expressive action language that speeds up the algorithm.

An independent proposal that also formulates planning as heuristic search is McDermott's [8]. The proposed system, UNPOP, uses a more efficient method for obtaining heuristic estimates (the bulk of the computation is shared by all the children of the same node) and does not require action schemas to be grounded. Yet the resulting heuristic estimates are not as informative, and the range of problems that can be solved and the quality of the solutions that are found do not appear to be as good as those obtained by HSP.

1.7 Current Work

Since the competition, we have reimplemented the code for computing the heuristic values. The resulting version of HSP runs several times faster and is available at `http://www.ldc.usb.ve/~hector`. We have also extended the ideas in [2] for planning in domains with probabilistic actions and noisy sensors as reported in [3].

References

[1] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, Montreal, Canada, 1995.

[2] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*, pages 714–719. MIT Press, 1997.

[3] H. Geffner and B. Bonet. High-level plannnig and control with incomplete information using POMDPs. In *Proceedings AIPS-98 Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, 1998.

[4] W. Harvey and M. Ginsberg. Limited discrepancy search. In *Proceedings IJCAI-95*, pages 607–613, 1995.

[5] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, pages 1194–1201, Protland, Oregon, 1996. MIT Press.

[6] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[7] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.

[8] D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proc. Third Int. Conf. on AI Planning Systems (AIPS-96)*, 1996.

[9] N. Nilsson. *Principles of Artificial Intelligence*. Tioga, 1980.

[10] J. Pearl. *Heuristics*. Morgan Kaufmann, 1983.