Heuristic Search Planner 2.0

Blai Bonet and Hector Geffner
Depto. de Computación
Universidad Simón Bolívar
Aptdo. 89000, Caracas 1080-A, Venezuela
{bonet,hector}@ldc.usb.ve

1 Introduction

HSP2.0 is a new version of the Heuristic Search Planner (HSP), a domain-independent planner that was entered into the first planning competition [14, 6]. These planners, as well as the regression planner HSPr [2] and the optimal planner HSPr* [7], are all based on the same idea: planning problems are mapped into search problems in a suitable space, which is solved using an heuristic function extracted automatically from the problem encoding. This formulation of planning as heuristic search appears in [12] and [4] (yet see also [10]). Other heuristic search planners in the AIPS2000 Contest are GRT [17], FF [8], and AltAlt [16]. Planners such as MIPS [5] and STAN [11] also make use of heuristic search ideas in certain contexts.

Pure heuristic search planners can be chacterized along three main dimensions: the state space that is searched (either the progression or regression space), the search algorithm used (most often some version of best-first or hill-climbing search), and the heuristic function extracted from the problem representation. The original version of HSP, for example, searches the progression space with a hill-climbing algorithm, and a non-admissible heuristic derived form a relaxation where delete lists are assumed empty and atoms are assumed independent. FF searches the same progression space using a different hill-climbing algorithm and a different non-admissible heuristic. Something similar is done by GRT. AltAlt, on the other hand, derives the heuristic from the plangraph constructed by a Graphplan type of procedure and uses this heuristic to drive a regression search from the goal. As a last example, HSPr* searches the regression space using the IDA* algorithm and an admissible heuristic function computed using a shortest-path algorithm.

Since no choice of the state space, search algorithm, and heuristic function appears best across different domains, we developed the HSP2.0 planner as a general platform for experimenting with different state spaces and different heuristics. The search algorithm in HSP2.0 is currently fixed and implements the WA* algorithm [15, 9], an A* algorithm in which the heuristic term $h(n)$ of the evaluation function $f(n) = g(n) + h(n)$ is multiplied by a parameter $W \geq 1$.

It is well known that the parameter W achieves a tradeoff between optimality and speed. For $W = 1$, WA* becomes A* and is optimal as long as the heuristic function is admissible (non-overestimating). For higher W, e.g., $W = 2$, WA* finds solutions faster but these solutions are only guaranteed to be at most W times away from optimal.

For a particular planning problem, the user of HSP2.0 can specify the state space to be searched, the heuristic, and the W parameter.[1] For example, the invocation

```
hsp -d backward -h h2max -w 1.5  prob4.pddl  domain.pddl
```

runs HSP over the regression space using the heuristic h^2 and the parameter $W = 1.5$, over the instance in file `prob4.pddl` with domain file `domain.pddl`. The syntax for the instance and domain files is given by the PDDL standard [13]. HSP2.0 deals with a fragment of PDDL that includes Strips, negation, types, and conditional effects.

In HSP2.0, the search can be done either forward or backward, and the heuristics can be the non-admissible heuristic h_{add} used in the original version of HSP, the admissible heuristic h_{max}, and the more informed admissible heuristic h^2 formulated in [7]. By default, HSP2.0 searches in the forward direction with $h = h_{add}$ and $W = 2$. The user can also choose a *schedule* of options as done in Blackbox. For example, it can tell HSP2.0 to use some direction and heuristic for a fixed amount of time, and then switch to a different direction/heuristic combination if no solution is found. In addition, the different options can be run *concurrently* as threads. This was actually the way HSP2.0 was run in the AIPS2000 Competition. HSP2.0 ran three options concurrently for three minutes: forward/h_{add}, backward/h_{add}, and backward/h^2, all with $W = 2$. If no solution was reported by then, HSP2.0 continued with the forward/h_{add} option only. The choice for the three minute threshold was empirical, following the observation that the regression search most often solved problems quickly or didn't solve them at all. The curve for the forward/h_{add} combination is smoother: more time, often means more problems solved. Some options clearly dominated others in particular domains: for example, in the `logistics` domain, most problems were solved by the backward/h_{add} option, while in the `FreeCell` domain most problems were solved by forward/h_{add}.

In the rest of this note, we will make more precise the options supported by the HSP2.0 planner, illustrate the optimality/speed tradeoff involved in the use of the W parameter, and briefly discuss some results.

2 State Spaces

A Strips problem is a tuple $P = \langle A, O, I, G \rangle$ where A is a set of atoms, O is a set of ground Strips operators, and $I \subseteq A$ and $G \subseteq A$ encode the initial and goal situations. The state space determined by P is a tuple $\mathcal{S} = \langle S, s_0, S_G, A(\cdot), f, c \rangle$ where

[1] HSP2.0 is available at `www.ldc.usb.ve/~hector` and `www.cs.ucla.edu/~bonet`.

S1. the states $s \in S$ are collections of atoms from A

S2. the initial state s_0 is I

S3. the goal states $s \in S_G$ are such that $G \subseteq s$

S4. the actions $a \in A(s)$ are the operators $op \in O$ such that $Prec(op) \subseteq s$

S5. the transition function f maps states s into states $s' = s - Del(a) + Add(a)$ for $a \in A(s)$

S6. the action costs $c(a)$ are all equal to 1

We refer to this state space as the *progression space*, in contrast to the *regression space* described below. When HSP is told to search in the forward direction, it searches the progression space starting from the initial state s_0.

The *regression state-space* associated with the same planning problem P can be defined by the tuple $\mathcal{R} = \langle \mathsf{S}, \mathsf{s}_0, \mathsf{S}_G, \mathsf{A}(\cdot), \mathsf{f} \rangle$ where

R1. the states $\mathsf{s} \in \mathsf{S}$ are sets of atoms from A,

R2. the initial state s_0 is the goal G,

R3. the goal states $\mathsf{s} \in \mathsf{S}_G$ are such $\mathsf{s} \subseteq I$,

R4. the set of actions $\mathsf{A}(\mathsf{s})$ applicable in s are the operators $op \in O$ that are *relevant* and *consistent;* namely, $Add(op) \cap \mathsf{s} \neq \emptyset$ and $Del(op) \cap \mathsf{s} = \emptyset$,

R5. the state $\mathsf{s}' = \mathsf{f}(a, \mathsf{s})$ that follows the application of $a = op$ in s, for $a \in \mathsf{A}(\mathsf{s})$, is such that $\mathsf{s}' = \mathsf{s} - Add(op) + Prec(op)$.

R6. the action costs $c(a)$ are all equal to 1

This is the state space that HSP2.0 searches when told to search *backward* from the goal. The regression search has the advantage that it makes it possible to compute the bulk of the heuristic function only once [2]. In the progression search, the heuristic values are computed from scratch in every new state. This recomputation is expensive, yet in certain cases it pays off by providing useful new information. Likewise, the regression search has the problem that it often generates spurious states: states that are not reachable from the initial problem situation such as those where one block is on top of two different blocks. Some of such states are detected by a procedure that identifies pairs of mutually exclusive propositions adapted from Graphplan [1]. Overall, the forward search is slower than the regression search in certain domains, but in general, it appears to be more robust [3].

3 Heuristics

The heuristics h_{add} and h_{max} in HSP are derived as approximations of the optimal cost function of a 'relaxed' planning problem in which delete lists are ignored. More precisely, the heuristics are obtained by combining the *estimated* *costs* $g(p; s)$ of achieving each of the goal atoms p from a state s. These estimates are obtained by solving the functional equation

$$g(p; s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } p \in s \\ \min_{a \in O(p)} [1 + g(Prec(a); s)] & \text{otherwise} \end{cases} \quad (1)$$

for all atoms p by means of a Bellman-Ford type of algorithm. In this algorithm the measures $g(p; s)$ are updated as

$$g(p; s) := \min_{a \in O(p)} [g(p; s), 1 + g(Prec(a); s)] \quad (2)$$

starting with $g(p; s) = 0$ if $p \in s$ and $g(p; s) = \infty$ otherwise, until they do not change. In these expressions, $O(p)$ stands for the set of operators that 'add' p and $g(Prec(a); s)$ stands for the estimated cost of achieving the *set* of atoms $Prec(op)$ from s.

For the additive heuristic h_{add}, the cost $g(C; s)$ of *sets* of atoms C is defined as the *sum* of the costs $g(r; s)$ of the individual atoms r in C. We denote such additive costs as $g_{add}(C; s)$:

$$g_{add}(C; s) \stackrel{\text{def}}{=} \sum_{r \in C} g(r; s) \quad \text{(additive costs)} \quad (3)$$

The heuristic $h_{add}(s)$ is then defined as:

$$h_{add}(s) \stackrel{\text{def}}{=} g_{add}(G; s) \quad (4)$$

The definition of the cost of *sets* of atoms in (3) assumes that 'subgoals' are *independent.* This is not true in general and as a result the heuristic may overestimate costs and is *not* admissible.

An admissible heuristic can be obtained by defining the costs $g(C; s)$ of sets of atoms as

$$g_{max}(C; s) \stackrel{\text{def}}{=} \max_{r \in C} g(r; s) \quad \text{(max costs)} \quad (5)$$

The resulting 'max' heuristic $h_{max}(s) = g_{max}(G; s)$ is admissible but it is not very informative. Both heuristics h_{add} and h_{max} are supported in HSP2.0.

The last heuristic supported in HSP2.0 is an admissible heuristic h^2 that dominates h_{max}. Both h^2 and h_{max} can be understood as instances of a general family of polynomial and admissible heuristics h^m for $m = 1, 2, \ldots$ [7]. The higher the value of m, the more accurate the heuristic but the more expensive its calculation. The idea underlying the heuristics h^m is to approximate the cost $g(C; s)$ for achieving a set of atoms C from a state s by the cost $g^m(C; s)$ of the most costly subset of size m in C. Thus, for $m = 1$, the cost $g(C; s)$ is approximated by the cost of the most costly atom in C, for $m = 2$, by the cost of the most costly atom pair in C, etc. The costs $g^m(C; s)$ are characterized by the equation

$$g^m(C; s) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } C \subseteq s, \text{ else} \\ \min_{B \in R(C)} [1 + g^m(B; s)] & \text{if } |C| \leq m \\ \max_{D \subset C, |D| = m} g^m(D; s) & \text{otherwise} \end{cases} \quad (6)$$

where $B \in R(C)$ if B is the result of regressing the set of atoms C through some action a. The heuristic function $h^m(s)$ is then defined as $g^m(G; s)$ where G is the goal. HSP2.0 accommodates the heuristics h^m for $m = 1$ and $m = 2$ only. Both are obtained by solving equation (6) with a Bellman-Ford type of shortest path algorithm.

4 Results

As mentioned above, in the competition, HSP2.0 was run with three concurrent options for three minutes: forward/h_{add}, backward/h_{add}, and backward/h^2. If no solution was found by then, only the option forward/h_{add} was allowed to continue. HSP2.0 solved all problems in most of the domains, except for the `scheduling` domain in which it solved the smallest instances only, and `blocks-world`, where it failed to solve some of the large instances. In many cases, like in `blocks-world`, the plans generated were long and far from optimal. Overall, the planner FF based on similar ideas did better and in certain cases significantly better. This seems due mostly to three factors: the search direction (forward search appears to be more robust than backward search), an heuristic more accurate than h_{add} (the only heuristic used by HSP2.0 in the forward direction), and a good rule for quickly discarding nodes without computing their heuristic values. These features, and others, discussed at more length in the article by Jorg Hoffman, appear to make FF a more powerful planner than HSP. A potential advantage of HSP is that it uses a more systematic search algorithm and that it can be used to generate optimal or arbitrarely close to optimal plans. Optimal plans are obtained for the settings $h = h^2$, $dir = backward$, and $W = 1$. At the same time, if a value of $W > 1$ is used, the resulting plans are known to be at most W times away from optimal and they are found much faster. As an optimal planner, HSP2.0 appears to be competitive with optimal (parallel) Graphplan and SAT planners such as STAN and Blackbox (see Table 1). At the same time, plans can be obtained much faster with a small degradation in plan quality by using a value of W slightly higher than 1 (see Table 2). This tradeoff is not so easily available in either Graphplan or SAT planners.

References

[1] A. Blum and M. Furst. Fast planning through planning graph analysis. In C. Mellish, editor, *Proceedings of IJCAI-95*, pages 1636–1642. Morgan Kaufmann, 1995.

[2] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proceedings of ECP-99*, pages 359–371. Springer, 1999.

	HSP		STAN		BLACKBOX	
problem	length	time	length	time	length	time
7-0	20	0.18	20	0.05	20	0.28
7-1	22	0.19	22	0.13	22	0.78
7-2	20	0.17	20	0.08	20	0.43
8-0	18	0.18	19	0.13	18	0.79
8-1	20	0.20	20	0.15	20	1.16
8-2	16	0.18	16	0.07	16	0.46
9-0	30	0.44	30	0.29	30	3.20
9-1	28	0.21	28	0.17	28	1.21
9-2	26	0.25	26	0.14	26	0.89
10-0	34	1.37	34	0.95	34	7.78
10-1	32	361.19	32	24.24	32	220.03
10-2	34	1.61	34	1.44	34	19.35
11-0	32	333.72	32	346.94	32	409.47
11-1	–	–	–	–	30	504.61
11-2	34	38.65	34	89.96	34	114.42
12-0	34	3.99	34	15.98	34	50.30
12-1	34	1.00	34	2.39	34	24.52

Table 1: Optimal Planning: Results over `block-world` instances from competition for some optimal planners. HSP2.0 results obtained with options $d = backward$, $h = h^2$, and $W = 1$. Time reported in seconds.

	HSP, $W = 1$		HSP, $W = 1.25$		HSP, $W = 1.75$	
problem	length	time	length	time	length	time
7-0	20	0.18	20	0.17	22	0.17
7-1	22	0.19	22	0.18	22	0.21
7-2	20	0.17	20	0.17	20	0.18
8-0	18	0.18	18	0.18	18	0.17
8-1	20	0.20	20	0.21	20	0.18
8-2	16	0.18	16	0.22	16	0.18
9-0	30	0.44	32	0.36	32	0.35
9-1	28	0.21	28	0.25	30	0.21
9-2	26	0.25	26	0.24	26	0.19
10-0	34	1.37	36	0.57	40	0.53
10-1	32	361.19	32	18.48	34	0.37
10-2	34	1.61	34	0.59	38	0.23
11-0	32	333.72	32	2.60	34	0.28
11-1	–	–	–	–	–	–
11-2	34	38.65	34	0.62	34	0.65
12-0	34	3.99	34	0.53	34	0.31
12-1	34	1.00	34	0.31	36	0.29

Table 2: Optimality/Speed Tradeoff. Results for HSP2.0 over same instances but with different W values

[3] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*. Special Issue on Heuristic Search. W. Zhang, R. Korf, and R. Dechter (Eds). To appear, 2000.

[4] 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.

[5] S. Edelkamp and M. Helmert. On the implementation of MIPS. In *Proc. AIPS Workshop on Model-Theoretic Approaches to Planning*, 2000.

[6] Derek Long (editor). The AIPS-98 Planning Competition. *Artificial Intelligence Magazine*, 21(2):13–34, 2000.

[7] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proc. AIPS-2000*, 2000.

[8] J. Hoffman and B. Nebel. The FF planning system: Fast plan generation through heuristic search. Submitted, 2000.

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

[10] P. Laborie and M. Ghallab. Planning with sharable resources constraints. In C. Mellish, editor, *Proc. IJCAI-95*, pages 1643–1649. Morgan Kaufmann, 1995.

[11] D. Long and M. Fox. The efficient implementation of the plan-graph. *JAIR*, 10:85–115, 1999.

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

[13] D. McDermott. PDDL – the planning domain definition language. Available at `http://ftp.cs.yale.edu/pub/mcdermott`, 1998.

[14] D. McDermott. The 1998 AI Planning Systems Competition. *Artificial Intelligence Magazine*, 21(2):35–56, 2000.

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

[16] Z. Nguyen R. Sanchez Nigenda and S. Kambhampati. AltAlt: Combining the advantages of graphplan and heuristic state search. Technical report, Arizona State University, 2000.

[17] I. Refanidis and I. Vlahavas. GRT: A domain independent heuristic for Strips worlds based on greedy regression tables. In *Proceedings of ECP-99*. Springer, 1999.