Planning as Heuristic Search: New Results

Blai Bonet and Héctor Geffner*

Depto. de Computación
Universidad Simón Bolívar
Aptdo. 89000, Caracas 1080-A, Venezuela
{blai,hector}@usb.ve

Abstract. In the recent AIPS98 Planning Competition, the HSP planner, based on a forward state search and a suitable domain-independent heuristic, showed that heuristic search planners can be competitive with state of the art Graphplan and Satisfiability planners. HSP solved more problems than the other planners but often took more time or produced longer plans. The bottleneck in HSP is the computation of the heuristic for every new state. In this paper, we reformulate HSP so that this problem is avoided. The new planner, that we call HSP-R, is based on the same ideas and heuristic as HSP, but searches backward from the goal rather than forward from the initial state. We show that this change allows HSP-R to compute the heuristic function only once. As a result, HSP-R can produce better plans, often in less time. For example, HSP-R solves each of the 30 logistics problems from Kautz and Selman in less than 3 seconds. This is two orders of magnitude faster than BLACKBOX. At the same time, in almost all cases, the plans are substantially smaller. HSP-R is also more robust than HSP as it visits a larger number of states, makes deterministic decisions, and relies on a single adjustable parameter than can be safely fixed for most domains. HSP-R, however, is not better than HSP across all domains and in particular, in the blocks world, it fails on problems that HSP solves. We also discuss the relation between HSP-R and Graphplan, and argue that Graphplan is also best understood as a heuristic search planner with a precise heuristic function and search algorithm.

Keywords: Strips Planning, Heuristic Search, Empirical Evaluation, Graphplan

* For correspondence between May and September 1999, please use: Héctor Geffner, Dept of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden.
1 Introduction

The last few years have seen a number of promising new approaches in planning. Most prominent among these are Graphplan [BF95] and Satplan [KS96]. Both work in stages by building suitable structures and then searching them. In Graphplan, the structure is a graph, while in Satplan, it is a set of clauses. Both planners have shown impressive performance and have attracted a good deal of attention. Recent implementations and significant extensions have been reported in [KNHD97, LF99, KS99, ASW98].

In the recent AIPS'98 Planning Competition [McD98], three out of the four planners in the Strips track, hsp [KNHD97], stan [LF99], and blackbox [KS99], were based on these ideas. The fourth planner, hsp [BG98], was a planner based on heuristic search [Nil80, Pea83]. In hsp, the search is assumed to be similar to the search in problems like the 8-puzzle, the sole difference being in the heuristic: while in problems like the 8-puzzle, the heuristic is assumed as given (e.g., as the sum of Manhattan distances), in planning it has to be extracted from the representation of the problem. hsp thus appeals to a simple scheme for computing the heuristic from Strips encodings and uses the heuristic to guide the search.

In the competition, hsp solved more problems than the other planners but it often took more time or generated longer plans. A critical component in hsp is the computation of the heuristic. While this computation is relatively fast for a single state, hsp has to compute it for every state visited. In many instances, more than 80\% of the planner's time is spent on that. This same problem occurs in a related heuristic planner, unpop [McD96].

In this paper, we show that this problem can be eliminated by a suitable change in the direction of the search. A new planner, that we call hsp-r, is based on the same ideas and heuristic as hsp, but searches backward from the goal rather than forward from the initial state. We show that this change allows hsp-r to compute the heuristic only once. As a result, hsp-r can compute better plans than hsp, often in less time. For example, hsp-r solves each of the 30 logistics problems from the blackbox distribution in less than 3 seconds. This is two orders of magnitude faster than blackbox. At the same time, in almost all cases, the plans obtained are substantially smaller. hsp-r is also more robust than hsp as it visits a larger number of states, makes deterministic decisions, and relies on a single adjustable parameter than can be fixed for most domains. Nonetheless, hsp-r is no better than hsp across all domains. For example, in the blocks world, hsp solves instances that hsp-r does not.

In the paper, we also discuss the relation between hsp-r and Graphplan. Graphplan has quickly become a popular planning framework, and while it's not difficult to understand how Graphplan works, it's not as simple to understand why it works as well in comparison with traditional planners (yet see [KLP97]). Here we argue that Graphplan is also best understood as an 'heuristic search planner' with a very precise heuristic function and search algorithm.

The rest of the paper is organized as follows. First, we review hsp and its heuristic function (Section 2). Then we present hsp-r (Section 3) and a number
of experiments (Section 4). Finally, we discuss the relation between \textsf{hsp-r} and Graphplan (Sect. 5) and conclude with a brief summary (Section 6).

2 HSP: Heuristic Search Planner

\textsf{HSP} deals with planning problems as problems of heuristic search. A Strips problem \( P = (A, O, I, G) \) is a tuple where \( A \) is a set of atoms, \( O \) is a set of operators, and \( I \subseteq A \) and \( G \subseteq A \) encode the initial and goal situations.\(^1\) The operators \( op \in O \) are all assumed grounded, and each has precondition, add, and delete lists denoted as \( \text{Prec}(op) \), \( \text{Add}(op) \), and \( \text{Del}(op) \) given by sets of atoms from \( A \).

A Strips problem \( P = (A, O, I, G) \) defines a state-space \( \mathcal{S} = \langle S, s_0, S_G, A(\cdot), f, c \rangle \) where

\begin{enumerate}
  \item the states \( s \in S \) are collections of atoms from \( A \)
  \item the initial state \( s_0 \) is \( I \)
  \item the goal states \( s \in S_G \) are such that \( G \subseteq s \)
  \item the actions \( a \in A(s) \) are the operators \( op \in O \) such that \( \text{Prec}(op) \subseteq s \)
  \item the transition function \( f \) maps states \( s \) into states \( s' = f(s, a) \), for \( a = op \in A(s) \), such that \( s' = s - \text{Del}(op) + \text{Add}(op) \)
\end{enumerate}

\textsf{HSP} searches this state-space starting from \( s_0 \) with the aid of an heuristic extracted from the Strips representation of the problem (see also [McD96,BLG97]).

2.1 Heuristic

The heuristic function \( h \) for a planning problem \( P \) is obtained by considering a ‘relaxed’ problem \( P' \) in which all delete lists are ignored. 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 \( P \) from any \( s \). The heuristics \( h(s) \) for solving \( P \) could thus be set to \( h'(s) \). However, solving the relaxed problem \( P' \) optimally, and hence obtaining \( h'(s) \), is still exponentially hard. We thus set for an approximation: we set \( h(s) \) to an estimate of the optimal value function \( h'(s) \) of the relaxed problem. These estimates are computed as follows.

For a given state \( s \), we create a new state \( s^* \) that is initialized to \( s \). Then, iteratively, we find all the operators \( op \) that are applicable in \( s^* \) and add the atoms \( p \in \text{Add}(op) \) to \( s^* \). Every time we add an atom \( p \) to \( s^* \), we update a measure \( g(p) \) that provides an estimate of the cost (number of steps) for achieving \( p \) from \( s \). For atoms \( p \in s \), this measure is initialized to \( 0 \), while for all other atoms, \( g(p) \) is initialized to \( \infty \). Then, when an operator \( op \) with preconditions \( C = \text{Prec}(op) \) that asserts \( p \) is applied, \( g(p) \) is updated as

\[
g(p) := \min \left[ g(p), 1 + g(C) \right]
\]

\(^1\) See [Lif86] and [BN95] among others for mathematical reconstructions of basic and extended versions of the original Strips framework [FN71].
where \( g(C) \) stands for the cost of achieving the conjunction of atoms \( C \) (see below). The expansions and updates continue until the measures \( g(p) \) do not change.

In hsp, the costs \( g(C) \) for conjunctions of atoms \( C \) is defined as the sum of the costs \( g(r) \) for \( r \in C \):

\[
g(C) = \sum_{r \in C} g(r) \quad \text{(additive costs)}
\]

The measures defined in this way, are not admissible\(^2\) but are in general quite informative.

The heuristic function \( h(s) \) that provides an estimate of the number of steps required to make the goal \( G \) true from state \( s \), is then defined as:

\[
h(s) = g(G)
\]

Note that the \( g \) measures depend on \( s \), which is the state from which they are computed. In hsp, these measures are recomputed from scratch in every new state \( s \) visited for computing \( h(s) \) following (3). This is expensive and takes most of the time of hsp.

The definition of the cost measures assumes that ‘subgoals’ are independent. In general, however, they are not, and as a result the heuristic function is not admissible (it may overestimate the true optimal costs). An admissible heuristic function could be defined by simply changing the formula (2) for computing the cost of conjunctions in (1) and (3) to

\[
g(C) = \max_{r \in C} g(r) \quad \text{(max costs)}
\]

The heuristic function that would result from this change is admissible but is less informative than the additive function defined above, and this is the reason that is not used in hsp. In Sect. 5, however, we will see that a refined version of such ‘max heuristic’ is at works in another planner where it is used to estimate ‘time steps’.

### 2.2 Algorithm

Provided with the heuristic \( h \), hsp approaches planning as a problem of heuristic search. While standard search algorithms like \( \ast \) or \( \text{IDA}\ast \) [Kor83] could be used, hsp uses a variation of hill-climbing. This is due to several reasons, the most important one being that the computation of \( h(s) \) for every new state is expensive. hsp thus tries to get to the goal with as few node evaluations as possible. Surprisingly, the hill-climbing search plus a few simple amendments, like a limited number of ‘plateau’ moves, restarts, and a memory to avoid past states, performs relatively well. We have obtained similar results before using

\(^2\) That is, the resulting costs \( g(p) \) may overestimate the true number of actions required for achieving \( p \) from \( s \).
Korf's \textit{lmta*} \cite{Kor90} in place of hill-climbing; see \cite{BLG97}. This is probably due to the fact that the heuristic $h$, while not admissible, is quite informative and most often drives the search in a good direction.

Table 1 from \cite{McD98}, shows the results for the Strips track of the recent AIPS Competition. As can be seen, \textit{hsp} solved more problems than the other planners but it often took more time or produced longer plans (see \cite{McD98} for details).

<table>
<thead>
<tr>
<th>Round</th>
<th>Planner</th>
<th>Avg. Time</th>
<th>Solved</th>
<th>Fastest</th>
<th>Shortest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round 1</td>
<td>BLACKBOX</td>
<td>1.49</td>
<td>63</td>
<td>16</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>HSP</td>
<td>35.48</td>
<td>82</td>
<td>19</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>IPP</td>
<td>7.40</td>
<td>63</td>
<td>29</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>STAN</td>
<td>55.41</td>
<td>64</td>
<td>24</td>
<td>47</td>
</tr>
<tr>
<td>Round 2</td>
<td>BLACKBOX</td>
<td>2.46</td>
<td>8</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>HSP</td>
<td>25.87</td>
<td>9</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>IPP</td>
<td>17.37</td>
<td>11</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>STAN</td>
<td>1.33</td>
<td>7</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

\begin{table}[h]
\centering
\begin{tabular}{lllll}
  \hline
  Round & Planner & Avg. Time & Solved & Fastest & Shortest \\
  \hline
  Round 1 & BLACKBOX & 1.49 & 63 & 16 & 55 \\
    & HSP & 35.48 & 82 & 19 & 61 \\
    & IPP & 7.40 & 63 & 29 & 49 \\
    & STAN & 55.41 & 64 & 24 & 47 \\
  Round 2 & BLACKBOX & 2.46 & 8 & 3 & 6 \\
    & HSP & 25.87 & 9 & 1 & 5 \\
    & IPP & 17.37 & 11 & 3 & 8 \\
    & STAN & 1.33 & 7 & 5 & 4 \\
  \hline
\end{tabular}
\caption{Results of the AIPS98 Competition (Strips) from McDermott. Columns show the number of problems solved by each planner, and the number of problems in which each planner was fastest or produced shortest plans.}
\end{table}

3 \textbf{HSP-R: Heuristic Regression Planning}

In \textit{hsp}, the bottleneck is the computation of the heuristic in every new state. The main innovation in \textit{hsp-r} is a simple change in the direction of the search that avoids this problem. As a result, the search proceeds faster, more states can be visited, and often better plans can be found in less time.

\textit{hsp-r} searches backward from the goal rather than forward from the initial state. This is an old idea in planning known as \textit{regression search} \cite{Nil80,Wei94}. In regression search, the states visited can be thought as \textit{subgoals}; namely, the ‘application’ of an action in a goal yields a situation in which the execution of the action would achieve the goal. The crucial point in \textit{hsp-r} is that since the initial state $s_0$ does not change during the search, the \textit{same} measures $g(p)$ that estimate the cost of achieving each atom $p$ from $s_0$ (Sect. 2.1) can be used to determine the cost of any situation obtained by regression from the goal. As a result, the measures $g(p)$ computed from $s_0$, do not have to be recomputed.

3.1 \textbf{State space}

\textit{hsp-r} and \textit{hsp} can be understood as searching two different state spaces. \textit{hsp} searches the \textit{progression space} $S$ defined by $[S1]$--$[S5]$ above, where the actions are the available Strips operators. \textit{hsp-r}, on the other hand, searches a \textit{regression space} $R$. This regression space is defined by $[R1]$--$[R5]$ above. However, the regression space is not the only change that \textit{hsp-r} ...)
The regression space \( R \) associated with a Strips problem \( \langle A, O, I, G \rangle \) can be defined, in analogy to the progression space \( S \) defined by [S1]–[S5] above, as the tuple \( \langle S, s_0, S_G, A(\cdot), f, c \rangle \) where

R1. the states \( s \) are sets of atoms from \( A \)
R2. the initial state \( s_0 \) is the goal \( G \)
R3. the goal states \( s \in S_G \) are such \( s \subseteq I \)
R4. the set of actions \( A(s) \) applicable in \( s \) are the operators \( op \in O \) that are \textit{relevant} and \textit{consistent}; namely, \( \text{Add}(op) \cap s \neq \emptyset \) and \( \text{Del}(op) \cap s = \emptyset \)
R5. the state \( s' = f(a, s) \) that follows the application of \( a = op \) in \( s \), for \( a \in A(s) \), is such that \( s' = s - \text{Add}(op) + \text{Prec}(op) \).

The solution of this state space is, like the solution of any space \( \langle S, s_0, S_G, A(\cdot), f, c \rangle \), a finite sequence of actions \( a_0, a_1, \ldots, a_n \) such that for a sequence of states \( s_0, s_1, \ldots, s_{n+1}, s_{i+1} = f(a_i, s_i) \), for \( i = 0, \ldots, n \), \( a_i \in A(s_i) \), and \( s_{n+1} \in S_G \). In addition, we demand that the reverse action sequence solves \( S \), as states can be generated in \( R \) that are not reachable from the initial state in \( S \) (see Sect. 3.3).

### 3.2 Heuristic

\textsc{hsp-r} searches the regression space \([R1]–[R5]\) using an heuristic based on the additive cost estimates \( g(p) \) used by \textsc{hsp}, described in Sect 2.1. \textit{These estimates are computed only once from the initial state} \( s_0 \in S \). This is the main distinction with \textsc{hsp}. The heuristic \( h(s) \) associated with any state \( s \) is then defined as

\[
h(s) = \sum_{p \in s} g(p)
\]

In other words, the heuristic \( h(s) \) that estimates the cost of reaching the goal from any \( s \) in \( R \) is set to the estimated cost of achieving \( s \) from the \textit{initial state} in \( S \).

### 3.3 Mutexes

The regression search often leads to states \( s \) that are not reachable from the initial state in \( S \) and cannot lead to a solution. Graphplan [BF95] recognized this problem and showed a way to deal with it based on the notion of \textit{mutual exclusivity relations} or \textit{mutexes}. In \textsc{hsp-r}, we use the same idea, under a slightly different formulation.

Roughly, two atoms \( p \) and \( q \) form a mutex \( \langle p, q \rangle \), when they cannot be both true in a state reachable from \( s_0 \). For example, in block problems, atoms like \( \text{on}(a, b) \) and \( \text{on}(a, c) \) will normally form a mutex. States containing mutex pairs can be safely pruned as they cannot participate in any solution. We are thus interested in recognizing as many mutexes as possible.

A tentative definition is to define a mutex \( \langle p, q \rangle \) as a pair of atoms that does not hold in \( s_0 \) and for which every action that asserts \( p \) deletes \( q \), and vice
versa, every action that asserts \( q \) deletes \( p \). This definition, however, is too weak, and does not recognize the pair \( \langle \text{on}(a, b), \text{on}(a, c) \rangle \) as a mutex, as actions like \( \text{move}(a, d, b) \) add the first atom but do not delete the second.

We thus use a different definition in which \( \langle p, q \rangle \) is recognized as mutex when the actions that add \( p \) and do not delete \( q \) still guarantee, through their preconditions, that \( q \) will not be true after the action. For that we identify sets of mutexes.

**Definition 1.** A set \( M \) of atom pairs \( \langle p, q \rangle \) is a mutex set given a set of operators \( O \) and an initial state \( s_0 \) iff for all \( \langle p, q \rangle \) in \( M \)

1. \( p \) and \( q \) are not both true in \( s_0 \),
2. for every \( op \in O \) that adds \( p \), \( q \in \text{Del}(op) \), or \( q \notin \text{Add}(op) \) and for some \( r \in \text{Prec}(op) \), \( \langle r, q \rangle \in M \), and
3. for every \( op \in O \) that adds \( q \), \( p \in \text{Del}(op) \), or \( p \notin \text{Add}(op) \) and for some \( r \in \text{Prec}(op) \), \( \langle r, p \rangle \in M \).

It is easy to verify that if \( \langle p, q \rangle \) belongs to a mutex set, then \( p \) and \( q \) are really mutually exclusive, i.e., no reachable state will contain them both. Also if \( M_1 \) and \( M_2 \) are two mutex sets, \( M_1 \cup M_2 \) will be a mutex set as well, and hence according to this definition, there is a single maximal mutex set. Rather than computing this maximal set however (that may be difficult), we compute an approximation as follows.

We say that \( \langle p, q \rangle \) is a 'bad pair' when it does not comply with one of the conditions 1–3 above. The procedure for approximating the largest mutex set starts with a set of pairs \( M_0 \), and iteratively removes from it all bad pairs until no bad pair remains. The resulting mutex set is denoted as \( M^* \). The initial set \( M_0 \) is chosen so that \( \langle p, q \rangle \) and \( \langle q, p \rangle \) are in \( M_0 \) iff for some \( op \in O \), \( p \in \text{Add}(op) \) and \( q \in \text{Del}(op) \).

A mutex in hsp-r will refer to a pair in \( M^* \). Like the analogous definition in Graphplan, the set \( M^* \) does not capture all actual mutexes, yet it's simple to compute and prunes a large part of the regression space.

### 3.4 Algorithm

The search algorithm in hsp-r searches the regression space \( \mathcal{R} \) using the heuristic \( h \) to guide the search, and pruning states that contain a mutex in \( M^* \). The choice of the actual algorithm follows from several considerations:

1. We want hsp-r to solve problem with large state-spaces (e.g., spaces of \( 10^{20} \) states are not uncommon).
2. Node generation in hsp-r, while faster than in hsp, is slow in comparison with domain-specific search algorithms.\(^3\)

\(^3\) HSP-R generates a few thousand nodes per second while Korf’s algorithms for the Rubik’s Cube and the 24-puzzle generate in the order of a million nodes per second [Kor98,KT96].
3. The heuristic function, while often quite informative, is not admissible and many times overestimates the true optimal costs.
4. State spaces in planning are quite redundant with many different paths leading to the same states.

These constraints are common to $\text{hsp}$ where node generation is several times slower. Slow node generation with large state-spaces rules out optimal search algorithms such as $\text{A}^*$, $\text{IDA}^*$, $\text{DFS}$ with Branch and Bound. $\text{hsp}$ uses instead Hill-Climbing with a few extensions like a memory of past states, a limited number of ‘plateau’ moves, and multiple restarts. In $\text{hsp-r}$, we wanted to take advantage of the faster node generation to use a more complete search algorithm. After some experimentation, we settled on a simple algorithm that we call $\text{Greedy Best First}$. $\text{gbfs}$ is a $\text{BFS}$ algorithm that uses a cost function $f(n) = g(n) + W \cdot h(n)$, where $g(n)$ is the accumulated cost (number of steps), $h(n)$ is the estimated cost, and $W \geq 1$ is a real parameter. $\text{gbfs}$ is that before selecting a node $n$ with minimum $f$-cost from open, $\text{gbfs}$ checks first whether a minimum $f$-cost child $n''$ of the last node $n'$ expanded is ‘good enough’. If so, $\text{gbfs}$ selects any such child; else it selects $n$ as $\text{BFS}$. In $\text{hsp-r}$, $n''$ is ‘good enough’ when $n''$ appears closer to the goal than its parent $n'$; i.e., when $h(n'') < h(n')$. $\text{gbfs}$ thus interleaves ‘Greedy’ and ‘$\text{BFS}$’ moves. In a way, $\text{gbfs}$ does Hill-Climbing but ‘backtracks’ in a $\text{BFS}$ mode, when a ‘discrepancy’ is found (where a discrepancy occurs when the heuristic is not improved). We also explored schemes in which such ‘discrepancies’ are counted and used to control a LDS [HG95] or pure $\text{BFS}$ algorithm, but didn’t get as good results. The results reported below for $\text{hsp-r}$ are all based on the $\text{gbfs}$ algorithm described above with $W = 5$. Small changes in $W$ have little effect in $\text{hsp-r}$, but some problems are not solved with $W = 1$.

4 Experiments

We report results on two domains for which there is a widely used body of difficult but solvable instances. We use the 30 logistic instances and the 5 largest block-world instances in the $\text{blackbox}$ distribution (that we refer to as $\text{log-i}$ and $\text{bw-k}$, $i = 1, \ldots, 30$, $k = 1, \ldots, 5$), as well as 5 hard block-world instances of our own (that we refer to as $\text{bw-i}$, $i = 6, \ldots, 13$). $\text{gbfs}$ is not to be confused with the cost measures $g(p)$ defined in Sect. 2.1 for atoms. $\text{gbfs}$ is a domain-independent planner (it uses control information tailored for each domain) but provides a reference for the results achievable in the domain. The results are shown in Table 2. The

---

4 Not to be confused with the cost measures $g(p)$ defined in Sect. 2.1 for atoms.
5 For the role of $W$ in $\text{BFS}$, see [Kor93].
6 The logistics instances are from the ‘logistics-strips’ directory; while the block-world instances are from the prodigy-bw directory (they contain 7, 9, 11, 15, and 19 blocks, respectively).
7 These instances contain 8, 10, 12, 14, and 16 blocks.
‘time’ column measures CPU time in seconds, while ‘steps’ displays the number of actions in the solutions found.\(^8\)

As the table shows, \textit{hsp-r} solves each of the 30 logistics problems in less than 3 seconds. This is faster (and less erratic) than \textit{hsp}, and two orders of magnitude faster than \textit{blackbox}. At the same time, in almost all cases, the plans produced by \textit{hsp-r} are substantially smaller than those produced by \textit{blackbox} and \textit{hsp}. The speed of \textit{hsp-r} is comparable indeed with \textit{tlplan} that includes fine-tuned knowledge for the domain. The plans obtained by \textit{tlplan}, however, are shorter (with one exception). The table also shows that the number of steps in optimal parallel plans can be far from the optimal number of steps. Thus if the concern is with the number of steps, there is no reason in principle to prefer optimal parallel planners to non-optimal sequential planners.

\textit{hsp-r}, however, is not better than \textit{hsp} across all domains. First, there are problems were neither planner does particularly well as a number of ‘grid’, ‘mystery’ and ‘mprime’ instances from the AIPS competition (see [McD98]). Moreover, \textit{hsp} performs very well on hard blocks-world problems, where it does better than \textit{hsp-r}. These results are shown in Table 3. Although we don’t fully understand yet when \textit{hsp} will run better than \textit{hsp-r}, the results suggest nonetheless that in many domains a bi-directional planner combining \textit{hsp-r} and \textit{hsp} could probably do better than each planner separately.

5 Related Work

\textit{hsp-r} and \textit{hsp} are descendants of the real-time planner \textit{asp} reported in [BLG97]. All three planners perform planning as heuristic search and use the same heuristic, but search in different directions or with different algorithms. Another planner based on similar ideas is \textit{unpop} [McD96]. Interestingly, while in \textit{hsp-r} the heuristic is derived by forward propagation and it’s used to guide a backward search, in \textit{unpop}, the heuristic is derived by backward propagation and it’s used to guide a forward search.

The two phases in \textit{hsp-r}, the forward propagation and the backward search, are in close correspondence with the two phases in Graphplan [BF95], where the graph is built by reasoning forward and is then searched backward. \textit{hsp-r} also builds on the notion of mutexes introduced by Graphplan. For the rest, \textit{hsp-r} and Graphplan look quite different. However, we argue below that Graphplan, like \textit{hsp-r}, is also best understood as an heuristic-search based planner with

\(^8\) The results for \textit{blackbox} and \textit{tlplan} were taken from [BK98]. The results for \textit{blackbox} in [BK98] are compatible with the results in the distribution but are slightly more detailed. From [BK98], \textit{blackbox} was run on a SPARC Ultra 2 with a 296MHz clock and 256M of RAM, while \textit{tlplan} was run on a Pentium Pro with a 200MHz clock and 128M of RAM. The results for \textit{hsp} and \textit{hsp-r} were obtained on a SPARC Ultra 5 running at 333MHz with 128M of RAM. In the case of \textit{hsp} and \textit{hsp-r}, individual problems are converted into C programs that are then compiled and run. This takes in the order of a couple of seconds for each instance. This time is not included in the table.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Time</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HSP</td>
<td>HSP</td>
</tr>
<tr>
<td>log-01</td>
<td>0.09</td>
<td>0.63</td>
</tr>
<tr>
<td>log-02</td>
<td>0.08</td>
<td>0.48</td>
</tr>
<tr>
<td>log-03</td>
<td>0.08</td>
<td>0.47</td>
</tr>
<tr>
<td>log-04</td>
<td>0.23</td>
<td>0.95</td>
</tr>
<tr>
<td>log-05</td>
<td>0.19</td>
<td>1.12</td>
</tr>
<tr>
<td>log-06</td>
<td>0.33</td>
<td>1.51</td>
</tr>
<tr>
<td>log-07</td>
<td>1.12</td>
<td>—</td>
</tr>
<tr>
<td>log-08</td>
<td>1.49</td>
<td>0.43</td>
</tr>
<tr>
<td>log-09</td>
<td>0.76</td>
<td>—</td>
</tr>
<tr>
<td>log-10</td>
<td>0.99</td>
<td>4.77</td>
</tr>
<tr>
<td>log-11</td>
<td>0.64</td>
<td>1.61</td>
</tr>
<tr>
<td>log-12</td>
<td>0.43</td>
<td>1.30</td>
</tr>
<tr>
<td>log-13</td>
<td>1.26</td>
<td>5.78</td>
</tr>
<tr>
<td>log-14</td>
<td>1.46</td>
<td>8.48</td>
</tr>
<tr>
<td>log-15</td>
<td>1.10</td>
<td>3.53</td>
</tr>
<tr>
<td>log-16</td>
<td>0.34</td>
<td>1.09</td>
</tr>
<tr>
<td>log-17</td>
<td>0.36</td>
<td>—</td>
</tr>
<tr>
<td>log-18</td>
<td>2.36</td>
<td>5.88</td>
</tr>
<tr>
<td>log-19</td>
<td>0.74</td>
<td>2.97</td>
</tr>
<tr>
<td>log-20</td>
<td>1.77</td>
<td>7.46</td>
</tr>
<tr>
<td>log-21</td>
<td>1.51</td>
<td>5.09</td>
</tr>
<tr>
<td>log-22</td>
<td>1.22</td>
<td>4.12</td>
</tr>
<tr>
<td>log-23</td>
<td>0.95</td>
<td>—</td>
</tr>
<tr>
<td>log-24</td>
<td>1.05</td>
<td>5.00</td>
</tr>
<tr>
<td>log-25</td>
<td>1.04</td>
<td>—</td>
</tr>
<tr>
<td>log-26</td>
<td>0.80</td>
<td>3.66</td>
</tr>
<tr>
<td>log-27</td>
<td>1.08</td>
<td>4.45</td>
</tr>
<tr>
<td>log-28</td>
<td>2.96</td>
<td>—</td>
</tr>
<tr>
<td>log-29</td>
<td>2.40</td>
<td>8.40</td>
</tr>
<tr>
<td>log-30</td>
<td>0.89</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 2. Performance over logistics problems. A dash indicates that the planner was not able to solve the problem in 10 mins or 100 Mb.
### Table 3. Performance over blocks-world problems. A dash indicates that a planner was not able to solve the problem in 10 mins or 100 Mb.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Blocks</th>
<th>Time Steps</th>
<th>HSP-R Time</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>bu-01</td>
<td>7</td>
<td>0.07</td>
<td>6</td>
<td>0.16</td>
</tr>
<tr>
<td>bu-02</td>
<td>9</td>
<td>0.15</td>
<td>6</td>
<td>0.59</td>
</tr>
<tr>
<td>bu-03</td>
<td>11</td>
<td>0.33</td>
<td>10</td>
<td>0.77</td>
</tr>
<tr>
<td>bu-04</td>
<td>15</td>
<td>8.66</td>
<td>18</td>
<td>5.88</td>
</tr>
<tr>
<td>bu-05</td>
<td>19</td>
<td>51.14</td>
<td>25</td>
<td>15.86</td>
</tr>
<tr>
<td>bu-06</td>
<td>8</td>
<td>0.12</td>
<td>19</td>
<td>0.42</td>
</tr>
<tr>
<td>bu-07</td>
<td>10</td>
<td>1.10</td>
<td>13</td>
<td>0.65</td>
</tr>
<tr>
<td>bu-08</td>
<td>12</td>
<td>3.73</td>
<td>14</td>
<td>1.97</td>
</tr>
<tr>
<td>bu-09</td>
<td>14</td>
<td>–</td>
<td>–</td>
<td>3.83</td>
</tr>
<tr>
<td>bu-10</td>
<td>16</td>
<td>–</td>
<td>–</td>
<td>12.63</td>
</tr>
</tbody>
</table>

a precise heuristic function and a standard search algorithm. From this point of view, the main novelty in Graphplan (that is not negligible at all) is the implementation of the search algorithm, that is quite efficient and smart, and the way the heuristic function is derived.

Basically, we argue that the main features of Graphplan can be understood as follows:

1. **Graph:** The graph encodes an admissible heuristic $h_G$ that is a refined version of the 'max' heuristic discussed in Sect. 2.1 (as opposed to the non-admissible 'additive' heuristic used in HSP-R), where $h_G(s) = j$ iff $j$ is the index of the first level in the graph that contains (the set of atoms) $s$ without a mutex, and in which $s$ is not memoized (note that memoizations areupdates on the heuristic function $h_G$; see 4).

2. **Mutex:** Mutexes are used to prune states in the regression search (as in HSP-R) and also to refine the 'max' heuristic (discussed in Sect. 2.1) into $h_G$.

3. **Algorithm:** The search algorithm is a version of Iterative Deepening A* (IDA*) [Kor93], where the sum of the accumulated cost $g(n)$ and the estimated cost $h_G(n)$ is used to prune nodes $n$ whose cost exceed the current threshold. Actually, in Graphplan, such nodes are never generated.

4. **Memoization:** Memoizations are updates on the heuristic function $h_G$ (see 1). The resulting algorithm is a memory-extended version of IDA* that corresponds to MREC [SB89, MI98]. In MREC, the heuristic of a node $n$ is updated and stored in a hash-table after the search below the children of $n$ completes without a solution (given the current threshold).

5. **Parallelism:** Graphplan searches a regression space $R_p$ that is slightly different than $R$ above, in which the actions are macro-actions (parallel-actions) composed of compatible primitive actions. Solutions costs in $R_p$ are given by the number of macro-actions used (the number of time steps). The heuristic $h_G(\cdot)$ is admissible in this space as well, where it provides a tighter lower bound than in $R$. While the branching factor in $R_p$ can be very high (the
number of macro-actions applicable in a state), Graphplan makes smart use of the information in the graph to generate only the children that are ‘relevant’ and whose cost does not exceed the current threshold.

This interpretation can be tested by extracting the heuristic \( h_G \) from the graph and plugging it into a suitable version of IDA* (MREC) running over the parallel-regression space \( R_p \). The same care as Graphplan should be taken for generating the applicable ‘parallel’ actions in a state, avoiding redundant effort. Our claim is that the performance of the resulting algorithm should be close to the basic Graphplan system [BF95], except for a small constant factor.

We haven’t done this experiment ourselves but hope others may want to do it. If the account is correct, it will show that Graphplan, like hsp and hsp-r, is best understood as an heuristic search planner. This would provide a different perspective to evaluate the strengths and limitations of Graphplan, and it may suggest ways in which Graphplan can be improved.

6 Summary

We have presented a reformulation of hsp that makes ‘planning as heuristic search’ more competitive with Graphplan and SAT planners. The reformulation involves a change in the direction of the search that avoids the recomputation of the heuristic in every new state. We have shown that the resulting planner hsp-r is two orders of magnitude faster than BLACKBOX over a set of logistics problems, producing at the same time better sequential plans. hsp-r, however, does not always improve on hsp, and we have seen that in large block-world instances hsp does better. We have also discussed the relation between hsp-r and Graphplan, and argued that Graphplan can be best understood also as an heuristic search planner.

The strengths and weaknesses of hsp and hsp-r suggest that a bi-directional search planner combining them both could perform better than each planner separately. Other open challenges are the developments of better heuristics and search algorithms, and more efficient implementations.

The code for hsp-r will be available at www.ldc.usb.ve/~hector over the next few days.

Acknowledgements: Part of this work was done while H. Geffner was visiting IRIT in Toulouse, France. He thanks J. Lang, D. Dubois, H. Prade, and H. Farreny, for their hospitality and for making the visit possible. Both authors also thank R. Korf for comments related to this work. B. Bonet is currently at UCLA, Los Angeles, under a USB-Conicit fellowship.

References


