Planning as Heuristic Search: New Results

Blai Bonet and Héctor Geffner[*]

Depto. de Computación
Universidad Simón Bolívar
Aptdo. 89000, Caracas 1080-A, Venezuela
{bonet,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 accross 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, 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 AIPS98 Planning Competition [McD98], three out of the four planners in the Strips track, IPP [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 accross *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 HSP-R and Graphplan (Sect. 5) and conclude with a brief summary (Section 6).

2 HSP: Heuristic Search Planner

HSP deals with planning problems as problems of heuristic search. A Strips problem $P = \langle A, O, I, G \rangle$ 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 $Prec(op)$, $Add(op)$, and $Del(op)$ given by sets of atoms from A. A Strips problem $P = \langle A, O, I, G \rangle$ defines a state-space $\mathcal{S} = \langle S, s_0, S_G, A(\cdot), f, c \rangle$ where

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 $s' = f(s, a)$, for $a = op \in A(s)$, such that $s' = s - Del(op) + Add(op)$

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 $p \in Add(op)$ 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 ∞. Then, when an operator op with preconditions $C = Prec(op)$ that asserts p is applied, $g(p)$ is updated as

$$g(p) := \min \left[\; g(p) \; , \; 1 + g(C) \; \right] \tag{1}$$

---

[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 $r \in C$:

$$g(C) \stackrel{\text{def}}{=} \sum_{r \in C} g(r) \quad \text{(additive costs)} \tag{2}$$

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:

$$h(s) \stackrel{\text{def}}{=} g(G) \tag{3}$$

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

$$g(C) = \max_{r \in C} g(r) \quad \text{(max costs)} \tag{4}$$

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 A* or IDA* [Kor93] 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 amendements, 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 LRTA* [Kor90] in place of hill-climbing; see [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 [McD98], shows the results for the Strips track of the recent AIPS Competition. As can be seen, HSP solved more problems than the other planners but it often took more time or produced longer plans (see [McD98] for details).

Round	Planner	Avg. Time	Solved	Fastest	Shortest
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

Table 1. 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

3   HSP-R: Heuristic Regression Planning

In HSP, the bottleneck is the computation of the heuristic in every new state. The main innovation in 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.

HSP-R searches backward from the goal rather than forward from the initial state. This is an old idea in planning known as *regression search* [Nil80,Wel94]. In regression search, the states visited can be thought as *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 HSP-R is that since the initial state s_0 does not change during the search, the *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   State space

HSP-R and HSP can be understood as searching two different state spaces. HSP searches the *progression space* $\mathcal{S}$ defined by [S1]–[S5] above, where the actions are the available Strips operators. HSP-R, on the other hand, searches a *regression*

space $\mathcal{R}$ where the actions correspond to 'reverse' application of the Strips oper-
ators. The regression space $\mathcal{R}$ associated with a Strips problem $P = \langle A, O, I, G \rangle$
can be defined, in analogy to the progression space $\mathcal{S}$ defined by [S1]–[S5] above,
as the tuple $\langle \mathsf{S}, \mathsf{s}_0, \mathsf{S}_G, \mathsf{A}(\cdot), \mathsf{f}, \mathsf{c} \rangle$ 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 $\mathsf{s} \in \mathsf{S}_G$ 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 $\mathsf{s}' = \mathsf{s} - Add(op) + Prec(op)$.

The solution of this state space is, like the solution of any space $\langle \mathsf{S}, \mathsf{s}_0, \mathsf{S}_G, \mathsf{A}(\cdot), \mathsf{f}, \mathsf{c} \rangle$,
a finite sequence of actions $a_0, a_1, \ldots, a_n$ such that for a sequence of states s_0,
$\mathsf{s}_1, \ldots, \mathsf{s}_{n+1}, \mathsf{s}_{i+1} = \mathsf{f}(a_i, \mathsf{s}_i)$, for $i = 0, \ldots, n$, $a_i \in \mathsf{A}(\mathsf{s}_i)$, and $\mathsf{s}_{n+1} \in \mathsf{S}_G$. In
addition, we demand that the reverse action sequence solves $\mathcal{S}$, as states can be
generated in $\mathcal{R}$ that are not reachable from the initial state in $\mathcal{S}$ (see Sect. 3.3).

3.2   Heuristic

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

$$h(\mathsf{s}) = \sum_{p \in \mathsf{s}} g(p)$$

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

3.3   Mutexes

The regression search often leads to states s that are not reachable from the
initial state in $\mathcal{S}$ and cannot lead to a solution. Graphplan [BF95] recognized
this problem and showed a way to deal with it based on the notion of *mutual*
exclusivity relations or *mutexes.* In 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
$on(a, b)$ and $on(a, c)$ 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 $\langle on(a,b), on(a,c) \rangle$ as a mutex, as actions like $move(a,d,b)$ 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, $q \in Del(op)$, or $q \notin Add(op)$ and for some $r \in Prec(op)$, $\langle r,q \rangle \in M$, and*
3. *for every $op \in O$ that adds q, $p \in Del(op)$, or $p \notin Add(op)$ and for some $r \in Prec(op)$, $\langle r,p \rangle \in M$.*

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

$$\langle p,q \rangle \text{ and } \langle q,p \rangle \text{ are in } M_0 \text{ iff for some } op \in O, p \in Add(op) \text{ and } q \in Del(op)$$

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 HSP where node generation is several times slower. Slow node generation with large state-spaces rules out optimal search algorithms such as A*, IDA*, DFS with Branch and Bound. 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 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 *Greedy Best First*. GBFS is a 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),[4] $h(n)$ is the estimated cost, and $W \geq 1$ is a real parameter.[5] The difference between GBFS and BFS is that before selecting a node n with minimum f-cost from OPEN, GBFS checks first whether a minimum f-cost child n'' of the last node n' expanded is 'good enough'. If so, GBFS selects any such child; else it selects n as BFS. In HSP-R, n'' is 'good enough' when n'' appears closer to the goal than its parent n'; i.e., when $h(n'') < h(n')$. GBFS thus interleaves 'Greedy' and 'BFS' moves. In a way, GBFS does Hill-Climbing but 'backtracks' in a 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 BFS algorithm, but didn't get as good results. The results reported below for HSP-R are all based on the GBFS algorithm described above with $W = 5$. Small changes in W have little effect in 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 BLACKBOX distribution (that we refer to as log-i and bw-k, $i = 1, \ldots, 30$, $k = 1, \ldots, 5$),[6] as well as 5 hard block-world instances of our own (that we refer to as bw-i, $i = 6, \ldots, 13$).[7]

In logistics, we compare HSP-R with HSP, BLACKBOX, and TLPLAN. TLPLAN [BK98], unlike HSP, HSP-R, and BLACKBOX, is not 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 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, HSP-R solves each of the 30 logistics problems in less than 3 seconds. This is faster (and less erratic) than HSP, and two orders of magnitude faster than BLACKBOX. At the same time, in almost all cases, the plans produced by HSP-R are substantially smaller than those produced by BLACKBOX and HSP. The speed of HSP-R is comparable indeed with TLPLAN that includes fine-tuned knowledge for the domain. The plans obtained by 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.

HSP-R, however, is not better than HSP accross 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, HSP performs very well on hard blocks-world problems, where it does better than HSP-R. These results are shown in Table 3. Although we don't fully understand yet when HSP will run better than HSP-R, the results suggest nonetheless that in many domains a bi-directional planner combining HSP-R and HSP could probably do better than each planner separately.

5   Related Work

HSP-R and HSP are descendants of the real-time planner 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 UNPOP [McD96]. Interestingly, while in HSP-R the heuristic is derived by forward propagation and it's used to guide a backward search, in UNPOP, the heuristic is derived by backward propagation and it's used to guide a forward search.

The two phases in 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. HSP-R also builds on the notion of mutexes introduced by Graphplan. For the rest, HSP-R and Graphplan look quite different. However, we argue below that Graphplan, like HSP-R, is also best understood as an heuristic-search based planner with

---

[8] The results for BLACKBOX and TLPLAN were taken from [BK98]. The results for BLACKBOX in [BK98] are compatible with the results in the distribution but are slightly more detailed. From [BK98], BLACKBOX was run on a SPARC Ultra 2 with a 296 MHz clock and 256M of RAM, while TLPLAN was run on a Pentium Pro with a 200 MHz clock and 128M of RAM. The results for HSP and HSP-R were obtained on a SPARC Ultra 5 running at 333 MHz with 128M of RAM. In the case of HSP and 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.

Problem	Time				Steps			
	HSP-R	HSP	BBOX	TLPLAN	HSP-R	HSP	BBOX	TLPLAN
log-01	0.09	0.63	0.57	0.26	27	39	25	25
log-02	0.08	0.48	95.97	0.28	28	31	31	27
log-03	0.08	0.47	98.99	0.24	29	27	28	27
log-04	0.23	0.95	130.74	1.37	67	58	71	51
log-05	0.19	1.12	231.93	1.10	51	53	69	42
log-06	0.33	1.51	321.27	1.91	69	60	82	51
log-07	1.12	—	264.04	5.54	81	—	96	70
log-08	1.49	9.43	317.42	6.84	82	170	110	70
log-09	0.76	—	1609.45	3.79	77	—	121	70
log-10	0.99	4.77	84.04	2.42	46	109	71	41
log-11	0.64	1.61	137.93	2.24	54	47	68	46
log-12	0.43	1.30	136.22	1.93	41	36	49	38
log-13	1.26	5.78	165.84	6.54	74	102	85	66
log-14	1.46	8.48	77.74	9.34	82	141	89	73
log-15	1.10	3.53	424.36	5.36	69	76	91	63
log-16	0.34	1.09	926.96	1.14	44	43	85	39
log-17	0.36	—	758.47	1.24	48	—	83	43
log-18	2.36	5.88	152.35	9.27	56	57	105	46
log-19	0.74	2.97	149.22	2.66	50	68	78	45
log-20	1.77	7.46	538.22	10.18	99	144	113	89
log-21	1.51	5.09	190.49	6.83	69	81	87	59
log-22	1.22	4.12	846.84	6.40	87	99	111	75
log-23	0.95	—	173.93	4.69	70	—	85	62
log-24	1.05	5.00	74.83	4.71	73	113	87	64
log-25	1.04	—	73.99	4.09	67	—	84	57
log-26	0.80	3.66	233.40	3.64	52	77	80	55
log-27	1.08	4.45	145.16	5.52	76	115	97	70
log-28	2.96	—	867.34	14.53	88	—	118	74
log-29	2.40	8.40	89.51	5.99	50	105	84	45
log-30	0.89	—	495.37	3.42	52	—	92	51

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.

Problem	Blocks	HSP-R		HSP	
		Time	Steps	Time	Steps
bw-01	7	0.07	6	0.16	6
bw-02	9	0.15	6	0.59	10
bw-03	11	0.33	10	0.77	12
bw-04	15	8.66	18	5.88	19
bw-05	19	51.14	25	15.80	28
bw-06	8	0.12	19	0.42	12
bw-07	10	1.10	13	0.65	12
bw-08	12	3.73	14	1.97	18
bw-09	14	–	–	3.83	17
bw-10	16	–	–	12.63	20

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.

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 $h_G(\mathsf{s}) = j$ 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 are updates 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 $\mathcal{R}_p$ that is slightly different than $\mathcal{R}$ above, in which the actions are macro-actions (parallel-actions) composed of compatible primitive actions. Solutions costs in $\mathcal{R}_p$ 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 $\mathcal{R}$. While the branching factor in $\mathcal{R}_p$ 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 $\mathcal{R}_p$. 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.

References

[ASW98]   C. Anderson, D. Smith, and D. Weld. Conditional effects in graphplan. In *Proc. AIPS-98*. AAAI Press, 1998.

[BF95]   A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, Montreal, Canada, 1995.

[BG98]   B. Bonet and H. Geffner.   HSP: Planning as heuristic search. http://www.ldc.usb.ve/~hector, 1998.

[BK98]   F. Bacchus and F. Kabanza.   Using temporal logics to express search control knowledge for planning, 1998.   Submitted. Available at http://www.lpaig.uwaterloo.ca/~fbacchus.

[BLG97]   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.

[BN95]   C. Backstrom and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[FN71]   R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971.

[HG95]   W. Harvey and M. Ginsberg. Limited discrepancy search. In *Proc. IJCAI-95*, 1995.

[KLP97]   S. Kambhampati, E. Lambrecht, and E. Parker. Understanding and extending Graphplan. In S. Steel and R. Alami, editors, *Proc. 4th European Conf. on Planning*, volume LNAI 1248. Springer, 1997.

[KNHD97] J. Koehler, B. Nebel, J. Hoffman, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In S. Steel and R. Alami, editors, *Proc. 4th European Conf. on Planning*, volume LNAI 1248. Springer, 1997.

[Kor90]   R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[Kor93]   R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.

[Kor98]   R. Korf. Finding optimal solutions to to Rubik's cube using pattern databases. In *Proceedings of AAAI-98*, pages 1202–1207, 1998.

[KS96]   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.

[KS99]   H. Kautz and B. Selman.   Unifying SAT-based and Graph-based planning.   In *Proceedings IJCAI-99*, 1999.   To appear. Available at http://www.research.att.com/~kautz/blackbox.

[KT96]   R. Korf and L. Taylor. Finding optimal solutions to the twenty-four puzzle. In *Proceedings of AAAI-96*, pages 1202–1207. MIT Press, 1996.

[LF99]   D. Long and M. Fox. The efficient implementation of the plan-graph. *JAIR*, 1999.

[Lif86]   V. Lifschitz. On the semantics of STRIPS. In *Proc. Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufmann, 1986.

[McD96]   D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proc. Third Int. Conf. on AI Planning Systems (AIPS-96)*, 1996.

[McD98]   D. McDermott.   AIPS-98 Planning Competition Results. http://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html, 1998.

[MI98]   T. Miura and T. Ishida. Stochastic node caching for memory-bounded search. In *Proc. AAAI-98*, pages 450–456, 1998.

[Nil80]   N. Nilsson. *Principles of Artificial Intelligence*. Tioga, 1980.

[Pea83]   J. Pearl. *Heuristics*. Morgan Kaufmann, 1983.

[SB89]   A. Sen and A. Bagchi. Fast recursive formulations for BFS that allow controlled used of memory. In *Proc. IJCAI-89*, pages 297–302, 1989.

[Wel94]   D. Weld. An introduction to least commitment planning. *AI Magazine*, 1994.