Admissible Heuristics for Optimal Planning

Patrik Haslum
Department of Computer
and Information Science
Linköping University
S-58183 Linköping, Sweden

Héctor Geffner
Departamento de Computación
Universidad Simón Bolívar
Aptdo 89000, Caracas 1080-A
Venezuela

Abstract

HSP and HSPr are two recent planners that search the state-space using an heuristic function extracted from Strips encodings. HSP does a forward search from the initial state recomputing the heuristic in every state, while HSPr does a regression search from the goal computing a suitable representation of the heuristic only *once.* Both planners have shown good performance, often producing solutions that are competitive in time and number of actions with the solutions found by Graphplan and SAT planners. HSP and HSPr, however, are not *optimal planners.* This is because the heuristic function is not admissible and the search algorithms are not optimal. In this paper we address this problem. We formulate a new *admissible* heuristic for planning, use it to guide an IDA* search, and empirically evaluate the resulting *optimal* planner over a number of domains.

The main contribution is the idea underlying the heuristic that yields not one but a whole family of polynomial and admissible heuristics that trade accuracy for efficiency. The formulation is general and sheds some light on the heuristics used in HSP and Graphplan, and their relation. It exploits the factored (Strips) representation of planning problems, mapping shortest-path problems in *state-space* into suitably defined shortest-path problems in *atom-space.* The formulation applies with little variation to sequential and parallel planning, and problems with different action costs.

Introduction

HSP and HSPr are two recent planners that search the state-space using an heuristic function extracted from Strips encodings (Bonet & Geffner 1999). HSP does a forward search from the initial state computing the heuristic in every state, while HSPr does a regression search from the goal, computing a suitable representation of the heuristic only once. Both planners have shown good performance, often producing solutions that are competitive in time and number of actions with the solutions found by Graphplan and SAT planners (McDermott 1998).

HSP and HSPr, however, are not *optimal planners.* This is because the heuristic is *not* admissible and the search algorithms are not optimal.[1] Graphplan (Blum & Furst 1995) and Blackbox (Kautz & Selman 1999) are optimal *parallel* planners that guarantee a minimal number of *time steps* in the plans found. While optimality is not always a main concern in planning, the distinction between optimal and non-optimal algorithms is relevant in practice and is crucial in theory where optimal and approximate versions of the same problem may belong to different complexity classes (Garey & Johnson 1979).

The goal of this paper is to address this issue. For this, we formulate a new domain-independent *admissible* heuristic for planning and use it for computing *optimal* plans. The new heuristic is simple and general, and can be understood as mapping the shortest-path (planning) problem in *state-space* into a suitably defined shortest-path problem in *atom-space.* This idea is implicit in a number of recent planners, e.g., (Blum & Furst 1995; McDermott 1996; Bonet, Loerincs, & Geffner 1997); here we make it explicit and general. The formulation applies with little variation to problems with different *action costs* and *parallel* actions, and suggests extensions for other classes of problems such as problems with actions with different durations (e.g., (Smith & Weld 1999)).

The new heuristic is based on computing admissible estimates of the costs of achieving *sets* of atoms from the initial state s_0. When the size of these sets is 1, the heuristic is equivalent to the h_{max} heuristic considered in (Bonet & Geffner 1999). When the size is 2, for *parallel planning,* the heuristic is equivalent to the heuristic implicit in Graphplan. The computation of the heuristic, however, does not build a layered graph nor does it rely on 'mutex relations'. On the other hand, its time and space complexity is polynomial in N^m, where N is the number of atoms in the problem and m is the size of the sets considered.

For the experiments in this paper, we use the heuristic that results from sets of size $m = 2$ (atom pairs). To

[1] A heuristic is not admissible when it may overestimate optimal costs, while a search algorithm is not optimal when it does not guarantee the optimality of the solutions found (Nilsson 1980; Pearl 1983).

avoid the recomputation of the heuristic in every state, we take the idea from HSPr and compute the heuristic once from the initial state and use it to guide a regression search from the goal.[2] The search is performed using the optimal algorithm IDA* (Korf 1985). We call the resulting optimal planner HSPr*. With the current implementation, HSPr* produces good results in sequential domains like Blocks World and the 8-puzzle, but weaker results on parallel domains like rockets or logistics. This is in contrast with the non-optimal HSPr planner that solves these problems very fast. We discuss these results, try to identify its causes, and draw some conclusions.

The paper is organized as follows. We cover first the relevant background including the heuristics used in HSP and Graphplan (Sect. 2). Then we introduce the new heuristic (Sect. 3), review the basic and enhanced version of the IDA* algorithm that we use (Sect. 4), and report results over a number of sequential domains (Sect. 5). Last we consider the extensions and results for parallel planning (Sect. 6) and close with a summary and discussion (Sect. 7).

Background

HSP

HSP maps Strips planning problems into problems of heuristic search (Bonet & Geffner 1999). 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 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. the states $s \in S$ are collections of atoms from A
2. the initial state s_0 is I
3. the goal states $s \in S_G$ are such that $G \subseteq s$
4. the actions $a \in A(s)$ are the operators $op \in O$ such that $Prec(op) \subseteq s$
5. the transition function f maps states s into states $s' = s - Del(a) + Add(a)$ for $a \in A(s)$
6. the action costs $c(a)$ are assumed to be 1

HSP searches this state-space, starting from s_0, with an heuristic function h derived from the Strips representation of the problem. A similar approach was used before in (McDermott 1996) and (Bonet, Loerincs, & Geffner 1997).

The heuristic h is derived as an approximation of the optimal cost function of a 'relaxed' problem P' in which *delete lists* are ignored. More precisely, $h(s)$ is obtained by adding up the *estimated costs* $g_s(p)$ for achieving each of the goal atoms p from s. These estimates are computed for *all* atoms p by performing incremental updates of the form

$$g_s(p) := \min_{a \in O(p)} [g_s(p), 1 + g_s(Prec(a))] \qquad (1)$$

starting with $g_s(p) = 0$ if $p \in s$ and $g_s(p) = \infty$ otherwise, until the costs $g_s(p)$ do not change. In (1), $O(p)$ stands for the set of operators that 'add' p and $g_s(Prec(a))$ stands for the estimated cost of the *set* of atoms in $Prec(op)$.

In HSP, the cost $g_s(C)$ of *sets* of atoms C is defined as the *sum* of the costs $g_s(r)$ of the individual atoms r in the set. We denote such cost as $g_s^{add}(C)$:

$$g_s^{add}(C) \stackrel{\text{def}}{=} \sum_{r \in C} g_s(r) \quad \text{(additive costs)} \qquad (2)$$

The heuristic $h(s)$ used in HSP, that we call $h_{add}(s)$, is then defined as:

$$h_{add}(s) \stackrel{\text{def}}{=} g_s^{add}(G) \qquad (3)$$

The definition of the cost of *sets* of atoms in (2) 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 $g_s(C)$ of sets of atoms as

$$g_s^{max}(C) = \max_{r \in C} g_s(r) \quad \text{(max costs)} \qquad (4)$$

The resulting 'max heuristic' $h_{max}(s) = g_s^{max}(G)$ is admissible but is not as informative as $h_{add}(s)$ and is not used in HSP. In fact, while the 'additive' heuristic combines the costs of *all* subgoals, the 'max' heuristic considers the most difficult subgoals only.

In HSP, the heuristic $h(s)$ and the atom costs $g_s(p)$ are computed from scratch in every state s visited. This is the main bottleneck in HSP and can take up to 85% of the computation time. For this reason, HSP relies on a form of hill-climbing search for getting to the goal with as few state evaluations as possible. Surprisingly this works quite well in many domains. In the AIPS98 Planning Contest, for example, HSP solved 20% more problems than the Graphplan and SAT planners (McDermott 1998). In many cases, however, the hill-climbing search finds poor solutions or no solutions at all.

HSPr

HSPr (Bonet & Geffner 1999) is a variation on HSP that removes the need to recompute the atom costs $g_s(p)$ in every state s. This is achieved by computing these costs *once* from the initial state and then performing a *regression* search from the goal.[3] In this search, the heuristic $h(s)$ associated with *any* state s is defined in terms of the costs $g(p) = g_{s_0}(p)$ computed from s_0 as

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

[2] The heuristic can also be used in the context of HSP. However, the overhead of computing the heuristic in every state does not appear to be cost-effective in general.

[3] Refanidis and Vlahavas propose a different way for avoiding these recomputations. Rather than calculating the heuristics by forward propagation and using it in a backward search, they compute the heuristic by backward propagation and use it to guide a forward search. See (Refanidis & Vlahavas 1999).

Since node evaluation in HSPr is faster than in HSP, HSPr uses a more systematic search algorithm that often produces better plans than HSP in less time.[4] For example, HSPr solves each of the 30 logistic problems in the BLACKBOX distribution in less than 3 seconds each (Bonet & Geffner 1999). HSPr, however, is not better than HSP across all domains as the information resulting from the recomputation of the heuristic in certain cases appears to pay off. In addition, the regression search often generates states that cannot lead to any solution as they violate basic invariants of the domain. To alleviate this problem, HSPr identifies atoms pairs that are unreachable from the initial state (atemporal mutexes) and prunes the states that contain them. This is an idea adapted from Graphplan.

Graphplan

Planning in HSPr consists of two phases. In the first, a forward propagation is used to compute the measures $g(p)$ that estimate the cost of achieving each atom from s_0, in the second, a regression search is performed using an heuristic derived from those measures. These two phases are in correspondence with the two phases in Graphplan (Blum & Furst 1995), where a plan graph is built forward in the first phase, and is searched backward in the second. As argued in (Bonet & Geffner 1999), the parallel between the two planners goes further. Graphplan can also be understood as an heuristic search planner based on precise heuristic function h_G and a standard search algorithm. The heuristic $h_G(s)$ is given by the index j of the first level in the graph that contains the atoms in s without a mutex, and the search algorithm is a version of Iterative Deepening A^* (IDA^*) (Korf 1985) where the *sum* of the accumulated cost and the estimated cost $h_G(n)$ is used to prune nodes n whose cost exceed the current threshold (actually Graphplan never generates such nodes).[5]

While Graphplan and HSPr can both be understood as heuristic search planners they differ in the heuristic and algorithms they use. In addition, HSPr is concerned with (non-optimal) sequential planning while Graphplan is concerned with (optimal) parallel planning.

A new admissible heuristic

HSP and HSPr can be used to find good plans fast but not provable *optimal* plans. This is because they are based on non-admissible heuristics and non-optimal search algorithms. For finding optimal plans, an admissible heuristic that can safely prune large parts of the search space is needed.

The non-admissible heuristic h_{add} used in HSP is derived as an *approximation* of the optimal cost function of a *relaxed* problem where deletes lists are ignored. This formulation raises two problems. First, the *approximation* is not very good as it ignores the *positive* interactions among subgoals that can make one goal simpler after a second one has been achieved (this results in the heuristic being non-admissible). Second, the *relaxation* is not good as it ignores the *negative* interactions among subgoals that are lost when delete lists are discarded. These two problems have been addressed recently in the heuristic proposed by Refanidis and Vlahavas (99). The proposed heuristic is more accurate but it is still non-admissible and largely ad-hoc. Here we aim to formulate an heuristic that addresses these limitations but which can be given a clear justification. The idea is simply to approximate the cost of achieving any set of atoms A from s_0 in terms of the estimated costs of achieving sets of atoms B of a suitable small size m. When $m = 1$, we approximate the cost of any set of atoms in terms of the estimated cost of the *atoms* in the set. When $m = 2$, we approximate the cost of any set of atoms in terms of the estimated cost of the *atom pairs* in the set, and so on. In the first case we will obtain the heuristic h_{max}; in the second, the Graphplan heuristic, etc. We make these ideas precise below.

The new heuristic is defined in terms of a relaxed problem, but the 'original' and 'relaxed' problems are formulated in a slightly different way than before. The original problem is seen now as a *single-source shortest-path problem* (Bertsekas 1995; Ahuja, Magnanti, & Orlin 1993). In a single-source shortest path problem one is interested in finding the shortest paths from a given source node to every other node in a graph. In our graph, the nodes are the states s, the (directed) links are the actions a that map one state into another, and the link costs are given by the action costs $c(a) > 0$. The source node is the initial state s_0, and the (directed) paths that connect s_0 with a state s correspond to the *plans* that achieve s from s_0.

A way to solve this shortest-path problem is by finding the optimal cost function V^* over the nodes s, where $V^*(s)$ expresses the cost of the optimal path that connects s_0 to s. This function V^* can be characterized as the solution of the Bellman equation:[6]

$$V^*(s) = \min_{\langle s',a\rangle \in R(s)} [c(a) + V^*(s')] \qquad (5)$$

where $V^*(s_0) = 0$ and $R(s)$ stands for the state-action pairs $\langle s', a\rangle$ such that a maps s' into s (i.e., $a \in A(s')$ and $s = f(a, s')$).

[4] The search algorithm in HSPr is complete but is not optimal. Optimal algorithms such as A^* are not used because they take more time and space, and since the heuristic is not admissible they still don't guarantee optimality.

[5] *Without* memoization, the search algorithm in Graphplan is standard IDA^*. *With* memoization, the search algorithm is a memory-extended version of IDA^* (Sen & Bagchi 1989; Reinfeld & Marsland 1994) where the heuristic of a node is updated and stored in a hash-table after the search beneath its children completes without a solution (given the current threshold).

[6] For V^* to be well-defined when some states are not reachable from s_0, it suffices to assume 'dummy' actions with infinite costs that connect s_0 with each state s.

The shortest-path problem defined by (5) can be solved by a number of algorithms resulting in a heuristic function V^* that perfectly estimates the distance of any state s from s_0. Of course, there are two problems with this idea: first, the solution of (5) is polynomial in $|S|$ but *exponential* in the number of atoms; and second, the function V^* cannot be used (directly) to guide a regression search from the goal. This is because the goal G does not denote a single state but a *set* of states s_G such that $G \subseteq s_G$. Thus for guiding a regression search from the goal, a cost function must be defined over *sets of atoms* A understood as representing the *set of states* that make A true.

So we turn to a slightly different shortest-path formulation defined over *sets of atoms* and let G^* stand for the optimal cost function in that space. For a set of atoms A, $G^*(A)$ stands for the optimal cost of achieving the set of atoms A from s_0 or alternatively, the optimal cost of achieving a state s where A holds. The equation characterizing the function G^* is

$$G^*(A) = \min_{\langle B,a\rangle \in R(A)} [c(a) + G^*(B)] \tag{6}$$

where $G^*(A) = 0$ if $A \subseteq s_0$ and $R(A)$ refers to the set of pairs $\langle B, a\rangle$ such that B is the result of regressing A through a. Formally, this set is given by the pairs $\langle B, a\rangle$ such that $A \cap Add(a) \neq \emptyset$, $A \cap Del(a) = \emptyset$, and $B = A - Add(a) + Prec(a)$. We call such set $R(A)$ the *regression set* of A.

In the new shortest-path problem the nodes are the possible sets of atoms A and each pair $\langle B, a\rangle$ in $R(A)$ stands for a directed link $B \rightarrow A$ in the graph with cost $c(a)$. Such links can be understood as expressing that A can be achieved by the action a from *any* state s where B holds. This shortest-path problem is not simpler than the problem (5) but has two benefits: first the function G^* can be used effectively to guide a regression search, and second, *admissible approximations* of G^* can be easily defined.

Let G stand for a function with the same domain as G^* and let's write $G \leq G^*$ if for any set of atoms A, $G(A) \leq G^*(A)$. It's simple to check that if G is the optimal cost function of a modified shortest path problem obtained by the *addition* of 'links', $G \leq G^*$ must hold. Likewise, $G \leq G^*$ must hold if links $B \rightarrow A$ are replaced by links $B' \rightarrow A$ of the same cost where B' is such that $G^*(B') \leq G^*(B)$. We can regard both modifications as 'relaxations' that yield cost functions G that are lower bounds on G^*.

With these considerations in mind, let's consider the relaxation of the shortest-path problem (6) where the links $B \rightarrow A$ for 'large' sets of atoms B, i.e., sets with size $|B| > m$ for some positive integer m, are replaced by links $B' \rightarrow A$ where B' is a *subset* of B with size $|B'| = m$. Since $B' \subset B$ implies $G^*(B') \leq G^*(B)$, it follows from the arguments above that the optimal cost function G^m of the resulting problem must be a lower bound on G^*.

This lower bound function G^m is characterized by the following equations:

$$G^m(A) = 0 \;\text{ if } A \subseteq s_0 \tag{7}$$

$$G^m(A) = \min_{\langle B,a\rangle \in R(A)} [c(a) + G^m(B)] \tag{8}$$

if $|A| \leq m$ & $A \not\subseteq s_0$, and

$$G^m(A) = \max_{B \subset A, |B| = m} G^m(B) \quad \text{if } |A| > m \tag{9}$$

For any positive integer m, the complexity of computing G^m is a low polynomial in the number of nodes (the number of atom sets A with size $|A|$ equal to or smaller than m) (Bertsekas 1995; Ahuja, Magnanti, & Orlin 1993). G^m is thus a polynomial and admissible approximation of the optimal cost function G^*. The approximation is based on defining the cost of 'large' sets of atoms A in Equation 9, in terms of the costs of its 'smaller' parts. Equations 7 and 8, on the other hand, are common to both G^m and G^*.

For any positive integer m, we define the heuristics h^m as

$$h^m(A) \stackrel{\text{def}}{=} G^m(A) \tag{10}$$

The heuristics h^m, for $m = 1, 2, \ldots$ are all admissible, and they represent different tradeoffs between accuracy and efficiency. Higher-order heuristics are more accurate but are harder to compute. For any fixed value of m, the computation of the heuristic h^m is a low polynomial in N^m, where N is the number of atoms. Below we consider the concrete form of these heuristics for $m = 1$ and $m = 2$. In both cases, we use the Strips representation of actions to characterize the regression set $R(A)$ in equation (8) which is the key equation defining the functions G^m.

The Max-atom heuristic

For $m = 1$, the heuristic h^m reduces to the heuristic h_{max} considered above. Indeed, for sets $A = \{p\}$ of size 1, the regression set $R(\{p\})$ is given by the pairs $\langle Prec(a), a\rangle$ for $a \in O(p)$, where $O(p)$ stands for the set of actions that 'add' p. As a result, equation (8) for G^m becomes

$$G^1(\{p\}) = \min_{a \in O(p)} [c(a) + G^1(Prec(a))]$$

The resulting shortest-path problem can be solved by a number of label-correcting algorithms (Bertsekas 1995; Ahuja, Magnanti, & Orlin 1993), in which estimates $g^1(\{p\})$ are updated incrementally as

$$g^1(\{p\}) := \min_{a \in O(p)} [g^1(\{p\})\ ,\ c(a) + g^1(Prec(a))]$$

until they do not change, starting with $g^1(\{p\}) = 0$ if $p \in s_0$ and $g^1(\{p\}) = \infty$ otherwise, Following (7) and (9), $g(\emptyset)$ is set to 0 and $g^1(A)$ for $|A| > 1$ is set to $\max_{p \in A} g^1(\{p\})$. When the updates terminate, the estimates g^1 can be shown to represent the function G^1 that solves equations (7-9) (Bertsekas 1995; Ahuja, Magnanti, & Orlin 1993). The complexity of these algorithms varies according to the order in which the updates are performed, yet it's always a low polynomial in the number of nodes (atoms sets A with size $|A| \leq m$).

The computation of the heuristic h_{max} described above corresponds to this procedure, and thus $h_{max} = h^1$. In other words, h_{max} is the heuristic obtained by approximating the cost of *sets of atoms* by the cost of *the most costly atom in the set.* The heuristic is admissible but is not sufficiently informative. The choice in HSP and HSPr was to approximate the cost of sets of atoms in a different way as *the sum of the costs of the atoms in the set.* This approximation yields an heuristic that is more informative but is not admissible. The option now is to consider the heuristics h^m for higher values of m.

The Max-pair heuristic

If we let $O(p\&q)$ refer to the set of actions that add both p and q, and $O(p|q)$ to the set of actions that add p but do not add or delete q, the equation (8) for $m = 2$ and $A = \{p, q\}$ becomes

$$\begin{aligned} G^2(\{p,q\}) = \min \{ & \min_{a \in O(p\&q)} [c(a) + G^2(Prec(a))]; \\ & \min_{a \in O(p|q)} [c(a) + G^2(Prec(a) \cup \{q\})]; \\ & \min_{a \in O(q|p)} [c(a) + G^2(Prec(a) \cup \{p\})] \} \end{aligned}$$

while the equation for $A = \{p\}$ becomes

$$G^2(\{p\}) = \min_{a \in O(p)} [c(a) + G^2(Prec(a))]$$

As before these equations can be converted into updates for computing the value of the function G^2 over all sets of atoms with size less than or equal to 2. This computation remains polynomial in the number of atoms and actions, and can be computed reasonably fast in most of the domains we have considered. We call the heuristic $h^2 = G^2$, the *max-pairs* heuristic to distinguish it from the *max-atom* heuristic h^1.

The consideration of *atom pairs* for the computation of the heuristic h^2 is closely related to the consideration of *mutex pairs* in the computation of the heuristic h_G in Graphplan. A distinction between h^2 and h_G is that the former is defined for *arbitrary action costs* and *sequential planning,* while the latter is defined for *unitary costs* and *parallel planning.* Later on, we will introduce a definition analogous to h^2 for *parallel* planning that is equivalent to Graphplan h_G.

Higher Order Heuristics

Equations 7–10 define a family of heuristics $h^m = G^m$ for $m \geq 1$. For each value of m, the resulting heuristic is admissible and polynomial, but the complexity of the sequence of heuristics h^m grows exponentially with m. The experiments we have performed are limited to h^m with $m = 2$. Certainly, it should be possible to construct domains where higher-order heuristics would be cost-effective but we haven't explored that. A similar situation exists in Graphplan with the computation of higher-order mutexes (Blum & Furst 1995). Higher-order heuristics may prove effective in complex domains like the 15-puzzle, Rubik, and Hanoi where subgoals interact in complex ways. The challenge is to compute such heuristics efficiently and use them with little overhead at run time.

Algorithms

Below we use the heuristic h^2 in the context of an IDA* search (Korf 1985). The IDA* algorithm consists of a sequence of depth-first searches extended with an heuristic function h and an upper bound parameter UB. During the search, nodes n for which the sum of the accumulated cost $g(n)$ and predicted cost $h(n)$ exceed the upper bound UB are pruned. Initially, UB is set to the heuristic value of the root node, and after a failed trial UB is set to the cost $g(n) + h(n)$ of the least-cost node that was pruned in that trial.

IDA* is guaranteed to find optimal solutions when the heuristic h is admissible, but unlike A* it is a linear-memory algorithm. Memory-enhanced versions of IDA* have been defined for saving time such as those relying on transposition tables (Reinfeld & Marsland 1994). In the experiments below we report the results of IDA* with and without transposition tables.

The performance of IDA* is often sensible to the order in which the children of a node are selected for expansion (this affects the last iteration of IDA*). In some of the experiments we use an arbitrary node ordering while in others we choose the ordering determined by the additive heuristic h_{add} from HSP.

Commutativity Pruning

In planning problems, it is common for different action sequences to lead to the same states. Linear-memory algorithms like IDA* do not detect this and may end up exploring the same fragments of the search space a number of times. This problem can often be alleviated by exploiting certain symmetries.

Let's say that two operators a and a' are *commutative* if neither one deletes atoms in the precondition or add list of the other, and that a *set* of actions is commutative when all the actions in the set are pairwise commutative. Commutative actions thus correspond to the actions that can be done in parallel in Graphplan or Blackbox, and can be recognized efficiently at compile time.

Clearly the order in which a set of commutative actions is applied is irrelevant to the resulting outcome. A simple way to eliminate the consideration of all orderings except *one,* is by imposing a fixed ordering '$\prec$' on all actions (e.g., see (Korf 1998)). A branch containing a contiguous sequence of commutative actions $a_1, a_2, \ldots, a_n$ is then accepted when it complies with this ordering (i.e., when $a_1 \prec a_2 \prec \cdots \prec a_n$) and is rejected otherwise. This means that a branch in the search tree can be *pruned* as soon as it contains a sequence of two consecutive commutative actions a_i, a_{i+1} such that $a_i \succ a_{i+1}$. We refer to this form of pruning as *commutativity pruning.*

Results

We call the planner obtained by combining the h^2 heuristic with the IDA* algorithm, HSPr*. HSPr* is an optimal sequential planner. The current implementation is in C. The results below were obtained on a Sun Ultra 10 running at 440 Mhz with 256 RAM. In the first experiments, we consider a number of (mostly) sequential domains and compare HSPr* with two state-of-the-art planners: STAN 3.0 (Long & Fox 1999) and BLACKBOX 3.6 (Kautz & Selman 1999). Both of these planners are optimal *parallel* planners, so they minimize the number of time steps but not necessarily the number of actions.

Table 1 shows results over instances from the blocks world, 8-puzzle, grid, and gripper domains. The formulation of the blocks-world is the one with the three 'move' actions. The notation `blocks-i` denotes an instance with i blocks. The 8-puzzle is a familiar domain (Nilsson 1980; Pearl 1983). The maximum distance between any two (reachable) configurations is 31. The grid and gripper instances correspond to those used in the AIPS Planning Contest (McDermott 1998).

In the table, $\#S$ and $\#A$ stand for the number of time steps and the number of actions in the plan. For sequential planners we report the number of actions while for parallel planners we report both.

The numbers in Table 1 show that over these domains the performance of HSPr* is comparable with STAN and slightly better than BLACKBOX. These numbers, however, are just an illustration as the planners can be run with a number of different options (STAN was run with the default options; BLACKBOX was run with the compact simplifier and the SATZ solver). An important difference between the three planners is the use of memory. STAN and BLACKBOX use of a lot of memory, and when they fail, most often is due to memory. In HSPr*, memory does not appear to be such a problem. In `grid-2`, for example, HSPr* ran for almost eight hours until it finally found an optimal solution. This is not good time performance, but illustrates the advantages of using linear memory. STAN proved superior to both HSPr* and BLACKBOX in the gripper domain where it apparently exploits some of the symmetries in the domain (Fox & Long 1999).

The results for HSPr* in these experiments were obtained using the three enhancements of IDA* discussed in the previous section: commutativity pruning , a transposition table with 10^5 entries, and node-ordering given by the heuristic h_{add}. These are general enhancements and most often they speed up the search. For testing this, we ran some experiments on the blocks world problems with all possible combinations of these enhancements. The results are shown in Table 2, where the number of nodes expanded (#N) and total time (T) are reported. While in the small problem, the enhancements make no difference, in the larger problem they do. However, the payoffs do not always add up; for example, commutative pruning (Com) cuts the run time significantly when used in isolation but makes little difference when node-ordering (Ord) and a transposition table (TT) are used.

Table 3 displays the quality of the heuristic h^2 in comparison with the optimal cost of the problem, and the time taken by the search with respect to the total time (that also includes the computation of the heuristic). It can be seen that the heuristic provides reasonable bounds in the block-world problems but poorer bounds in the other domains. In the 8-puzzle, the heuristic seems to be weaker than the domain-dependent Manhattan distance heuristic but we haven't made a detailed comparison. In most domains, the time for computing the heuristic is small when compared with the search time. The exception is the grid domain where the computation of the heuristic takes most of the time.

We have tried to run HSPr* over standard *parallel domains* like logistics and rockets but after many hours we didn't obtain any results. The most important cause for this is that for those domains the heuristic h^2, which estimates serial cost, is a poor estimator. In parallel domains, there are many independent subgoals, and in that case the additive heuristic h_{add} produces better estimates. Indeed, the non-optimal HSPr planner that uses the h_{add} heuristic solves these problems very fast (Bonet & Geffner 1999).[7] The admissible heuristics h^m defined in Sect. 3, however, can be modified so that they estimate *parallel* rather than *serial* cost. In that case, the estimates are tighter and can be used to compute optimal *parallel* plans.

Optimal Parallel Planning

Heuristics for Parallel Planning

The definition of the heuristics h^m can be modified to estimate *parallel* rather than *serial* costs by simply changing the interpretation of the regression set $R(A)$ appearing in the equation (8). This equation characterizes the cost function $G^m(A)$ for the sets $A \not\subseteq s_0$ and $|A| \leq m$ and is reproduced here

$$G^m(A) = \min_{\langle B,a\rangle \in R(A)} [c(a) + G^m(B)] \qquad (11)$$

Recall that $R(A)$ contains the pairs $\langle B, a\rangle$ such that B is the result of regressing A through action a. For making $h^m = G^m$ an estimator of parallel cost, all we need to do is to let a range over the set of *parallel actions,* where a parallel action stands for a set of pairwise compatible (commutative) actions.

We illustrate the result of this change for $m = 2$. We denote by G_p^m the cost function associated with the

[7] The reason for this, however, is not only the heuristic but also the search algorithm. The non-optimal search algorithm in HSPr can reach the goal by evaluating much fewer nodes than IDA*.

	STAN			BLACKBOX			HSPr*	
Instance	Time	#S	#A	Time	#S	#A	Time	#A
blocks-9	0.4	4	10	1.0	4	11	0.45	6
blocks-11	2.0	5	14	6.9	5	15	1.29	9
blocks-15	102.7	8	25	—	—	—	136.46	14
eight-1	89.9	31	31	—	—	—	63.53	31
eight-2	62.0	31	31	—	—	—	67.23	31
eight-3	0.5	20	20	89.0	20	20	0.51	20
grid-1	1.5	14	14	14.2	14	14	8.44	14
grid-2	—	—	—	—	—	—	7:55h	26
gripper-1	0.0	3	3	0.0	3	3	0.07	3
gripper-2	0.0	7	9	0.3	7	9	0.11	9
gripper-3	0.1	12	15	55.2	11	15	30.17	15
gripper-4	2.1	17	21	*	*	*	*	*

Table 1: Performance comparison over sequential domains. A long dash (—) indicates that the planner exhausted the available memory and a star (*) indicates that no solution was found after 12 hours. All times are in seconds.

Options			blocks-11		blocks-15	
Ord	TT	Com	#N	Time	#N	Time
off	off	off	1068	6.44	134680	3458.96
off	off	on	640	3.65	30145	566.50
off	on	off	480	3.17	29831	876.27
off	on	on	466	2.95	11255	236.56
on	off	off	137	1.17	51241	1280.82
on	off	on	96	0.93	30069	591.73
on	on	off	93	0.98	6280	165.25
on	on	on	87	0.93	7580	159.12

Table 2: Effects of IDA* enhancements in the number of nodes expanded and time taken by HSPr*. Time in seconds.

Instance	Opt.	$h(root)$	Nodes	Time	Search
blocks-9	6	5	9	0.45	0.02
blocks-11	9	7	87	1.29	0.42
blocks-15	14	11	6630	136.46	132.45
eight-1	31	15	172334	63.53	63.34
eight-2	31	15	182195	67.23	67.06
eight-3	20	12	564	0.51	0.20
grid-1	14	14	14	8.44	0.08
gripper-1	3	3	3	0.07	0.00
gripper-2	9	4	275	0.11	0.02
gripper-3	15	4	371664	30.17	30.06

Table 3: Results for sequential problems displaying optimal and estimated costs, expanded nodes, and total vs. search time. Time in seconds.

parallel problem, and let $O(p, q)$ stand for the set of compatible *pairs* of actions a and a' such that p and q belong to $Add(a) \cup Add(a')$. We assume now that all primitive and *parallel* actions have uniform cost $c(a) = 1$.[8] The definition of G^2_p then takes the form:

$$\begin{aligned} G^2_p(\{p,q\}) = \min \{ & \min_{a \in O(p\&q)} [1 + G^2_p(Prec(a))]; \\ & \min_{\langle a,a'\rangle \in O(p,q)} [1 + G^2_p(Prec(a) \cup Prec(a'))]; \\ & \min_{a \in O(p|q)} [1 + G^2_p(Prec(a) \cup \{q\})]\} \\ & \min_{a \in O(q|p)} [1 + G^2_p(Prec(a) \cup \{p\})]\} \end{aligned}$$

where the only change from the definition of *serial* G^2 is in the second line: the *parallel* action $a\&a'$ is allowed to establish the pair of atoms $p\&q$ at the cost of a primitive action. The equations for $G^2_p(A)$ for sets with size $|A| \neq 2$ remain the same as before. The resulting heuristic $h^2_p = G^2_p$, unlike the heuristic h^2, is *admissible* for parallel planning. Actually h^2_p can be shown to be equivalent to the heuristic h_G used in Graphplan where $h_G(s)$ stands for the first layer in the plan graph that includes the atoms in s without a mutex. For proving this, it is sufficient to show that h_G complies with the equations for G^2_p, and this can be done inductively starting with layer 0.

[8] It's not clear what the cost of a parallel action should be when primitive actions have different costs.

State Space for Parallel Planning

The simplest way to use the heuristic h^2_p to find optimal parallel plans is by performing a regression search from the goal with an algorithm like IDA* but replacing the *primitive* actions with the possible *parallel actions.* The problem with this idea, however, is that it does not scale up; indeed, if the branching factor of the original problem is b, the branching factor of the 'parallel' problem may be 2^b. While the solution length in the parallel space will be smaller, the growth in the branching factor makes the scheme impractical.

A second approach is to retain the branching structure from the serial setting but change the cost structure. The cost of an action a in the serial setting is normally uniform. In the parallel setting, it can be defined in terms of the preceding actions. The result is that total cost will measure time steps rather than action occurrences. This can be achieved by setting the cost of an action to 0 when the action is compatible with the 'last' actions in the search tree, and to 1 otherwise (the 'last' actions defined in a suitable way). The problem with this space is that it makes the heuristic h_p^2 not admissible. Admissibility can be restored by subtracting 1 from the value of the heuristic yet this transformation makes the heuristic much less powerful.

We have thus settled on a third alternative for finding optimal parallel plans that follows the scheme used in Graphplan. The resulting search space can be characterized as follows:

States: the states are triples $\langle Old, New, Acts \rangle$, where Old and New are sets of atoms, and $Acts$ is a set of pairwise compatible primitive actions.

Branching: the children of a state $\langle Old, New, Acts \rangle$ are obtained by applying all the primitive actions a that add the first atom p in Old and are compatible with all the actions in $Acts$. For each such action, the resulting state is $\langle Old - A(a), New + P(a), Acts + \{a\} \rangle$, where $P(a)$ and $A(a)$ stand for the precondition and add list of a respectively.

No-Ops: actions $No\_Op(p)$ with precondition and add list equal to p are assumed for each atom p

Costs: a dummy action that is the sole action applicable in the states $\langle \emptyset, A, Acts \rangle$ is assumed. Such action has cost 1 and leads to the state $\langle A, \emptyset, \emptyset \rangle$. All other actions have cost 0.

Heuristic: the heuristic of a state $\langle Old, New, Acts \rangle$ is given by $h_p^2(New)$, which is non-overestimating.

Init and Goal: the initial state of the regression is $\langle \emptyset, G, \emptyset \rangle$, where G is the goal, and the goal states are $\langle \emptyset, A, Acts \rangle$ for $A \subseteq s_0$, where s_0 is the initial situation.

In relation to Graphplan, the set of atoms Old in the state $\langle Old, New, Acts \rangle$ can be thought as the list of atoms in layer i that haven't been regressed yet, New stands for the atoms in layer $i-1$ that have been obtained from the regression so far, and $Acts$ encodes the actions that have been used to obtain those atoms.

We will refer to the planner that results from the use of the IDA* search over this space, *parallel* HSPr^* or HSPr$(p)^*$. Below we report results of this planner over some standard parallel domains and compare it with two state-of-the-art parallel planners and the original version of Graphplan.

HSPr$(p)^*$ has three main aspects in common with Graphplan: the heuristic, the search space, and the search algorithm. On the other hand, HSPr$(p)^*$ does not use a plan graph. The plan graph plays two roles in Graphplan. First, and most important, it encodes the heuristic. This aspect is captured by the use of the h_p^2 heuristic in HSPr$(p)^*$. However, the plan graph also stores information that makes the IDA* search more efficient: it makes regressions faster, it never generates nodes that will be pruned, etc. Indeed, the IDA* search in Graphplan takes the form of a 'solution extraction' algorithm in the plan graph. This second rol of the plan graph is not captured in HSPr$(p)^*$. On the positive side, HSPr$(p)^*$ requires less memory and can easily be modified to use other search algorithms such as A* or WIDA* (Korf 1993). Such changes can be accommodated in Graphplan but provided the plan graph is used mainly for representing and computing the heuristic and not for solution extraction.

Results for Parallel Planning

Table 4 shows results over some standard parallel domains. On the 'rocket' problems, HSPr$(p)^*$ appears to be slightly better than Graphplan, while in the 'logistics' problems, Graphplan is definitely superior. These differences are likely due to the use of the plan graph. As the columns for STAN and BLACKBOX show, neither Graphplan or HSPr$(p)^*$ are state-of-the-art over these domains. Nonetheless, STAN is a Graphplan-based planner that solves the logistics problems quite fast.[9]

To further compare the speed of HSPr$(p)^*$ and Graphplan we generated approximately 45 medium-sized, random logistics instances solvable by both HSPr$(p)^*$ and Graphplan. For the reasons above, we didn't expect HSPr$(p)^*$ to approach the speed of Graphplan but we did expect HSPr$(p)^*$ to remain within an order of magnitude. In 30 of the problems, that was the case. However, in 12 problems we found HSPr$(p)^*$ to be from 10 to 75 times slower than Graphplan, and in 3 problems we found HSPr$(p)^*$ to be between 75 and 200 times slower. These differences in speed are probably not only due to the use of the planning graph in the search but also to the node ordering used in both planners. Graphplan, for example, tries No-Op actions first, while HSPr$(p)^*$ tries them last. Similarly, in HSPr$(p)^*$ we have found it convenient to order the atoms in Old in the state $\langle Old, New, Acts \rangle$ by increasing value of the additive heuristic h_{add}. While these choices help in a number of examples, they also hurt in others, and thus potentially amplify the differences in performance over some of the instances.

Discussion

In this paper we have formulated a framework for deriving polynomial admissible heuristics for sequential and parallel planning, and have evaluated the performance of the optimal planner that results from using one of these heuristics with the IDA* algorithm. The work sheds light on the heuristics used in HSP and Graphplan, and provides a more solid basis for pursuing the

[9]For some reason, STAN didn't solve the rocket problems. Apparently, this is a bug that will be fixed.

Problem	HSP$r(p)^*$	GRAPHPLN	STAN	BBOX
rocket.a	90.5	100.0	—	1.8
rocket.b	68.6	310.0	—	2.3
log.a	3:12:20	0:20:35	0.4	2.1
log.b	*	0:9:39	2.0	10.4
log.c	*	—	0:21:01	47.0

Table 4: Time comparison over parallel domains. A long dash (—) indicates that the planner exhausted the available memory and a star (*) indicates that no solution was found after 12 hours. In the notation $h : m : s$, h, m and s stand for hours, minutes, and seconds respectively. Otherwise, times are in seconds.

'planning as heuristic search' approach. Below we discuss briefly related work and some open problems.

Graphplan: In (Bonet & Geffner 1999), Graphplan was described as an heuristic search planner based on an IDA* search and a heuristic $h_G(s)$ given by the first layer in the plan graph that contains the atoms in s without a mutex. In this paper, we have taken this view further, providing an explanation and a generalization of that heuristic, and evaluating a pure IDA* planner with respect to Graphplan. In Graphplan, the plan graph plays two roles: it's used for computing and representing the heuristic, and for making the IDA* search more efficient. These uses explain the efficiency of Graphplan in comparison to previous planners. On the othe hand, it's not clear whether the plan graph will be suitable for computing and representing higher order heuristics (h^m, for $m > 2$) and searching with other algorithms.

Heuristics: higher-order heuristics may prove effective in domains like the 15-puzzle, Hanoi, Rubik, etc, where subgoals interact in complex ways. The challenge is to compute such heuristics fast enough and to use them with little overhead at run-time. Higher-order (max) heuristics as defined in this paper are related to the heuristics based on *pattern databases* defined in (Culberson & Schaeffer 1998). Korf and Taylor (96) discuss ways for generating *hybrid* heuristics involving both 'max' and 'additive' operations that may also prove useful in planning.

Algorithms: the heuristics defined in this paper have been used in the context of the IDA* algorithm. In a number of domains, however, a best-first search may prove more convenient. When optimality is not an issue, variations of A* and IDA* where the heuristic is multiplied by a constant $W > 1$ may speed up the search considerably (Korf 1993), making the resulting planner competitive with the HSP and HSPr planners over domains like Hanoi and Tire-world, where the additive heuristic is not adequate.

Branching: in highly parallel domains like rockets and logistics, SAT approaches appear to do best. This may be due to the branching scheme used (see (Rintanen 1998)). In SAT formulations, the space is explored by setting the value of any variable at any time point, and then considering each of the resulting state partitions separately. In heuristic search approaches, the splitting is commonly done by applying all possible actions. Alternative branching schemes, however, are common in heuristic branch-and-bound search procedures (Lawler & Rinnooy-Kan 1985), and they may prove relevant in planning.

We hope to explore some of these ideas in the future.

Acknowledgments

We thank Blai Bonet for useful discussions on the topic of this paper. Part of this work was done while H. Geffner was visiting Linköping University. He thanks E. Sandewall and P. Doherty for making this visit possible and enjoyable. This work has been partially supported by grant S1-96001365 from Conicit, Venezuela and by the Wallenberg Foundation, Sweden. P. Haslum is also funded by the ECSEL/ENSYM Graduate Study Program.

References

Ahuja, R.; Magnanti, T.; and Orlin, J. 1993. *Network Flows*. Prentice-Hall.

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control, Vols 1 and 2.* Athena Scientific.

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

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *Proceedings of ECP-99*. Springer.

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

Culberson, J., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):319–333.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning domains. In *Proc. IJCAI-99*.

Garey, M., and Johnson, D. 1979. *Computers and Intractability.* Freeman.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and Graph-based planning. In *Proceedings IJCAI-99*.

Korf, R. 1985. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

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

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

Lawler, E., and Rinnooy-Kan, A., eds. 1985. *The Traveling Salesman Problem : A Guided Tour of Combinatorial Optimization.* Wiley.

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

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

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

Nilsson, N. 1980. *Principles of Artificial Intelligence*. Tioga.

Pearl, J. 1983. *Heuristics*. Morgan Kaufmann.

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

Reinfeld, A., and Marsland, T. 1994. Enhanced iterative-deepening search. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 16(7):701–710.

Rintanen, J. 1998. A planning algorithm not based on directional search. In *Proceedings KR'98*, 617–624. Morgan Kaufmann.

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

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. IJCAI-99*.