# Improving Heuristics through Search

**P@trik Haslum**[1]

**Abstract.** We investigate two methods of using limited search to improve admissible heuristics for planning, similar to pattern databases and pattern searches. We also develop a new algorithm for searching AND/OR graphs.

## 1 Introduction

In past work [5, 6] we have investigated a method of automatically deriving admissible heuristics for variations of classical planning. The main result is a family of heuristics, $H^m$, where $m$ represents a trade-off between accuracy and efficiency. For small $m$, $H^m$ can be computed quickly by a dynamic programming method, but the heuristic is often too weak, causing the planner to spend too much time in search. In this paper we investigate a different approach, in which we redirect some of the search effort from the search for a solution to improving cost estimates, by computing parts of $H^m$ for higher values of $m$.

## 2 Background

This section briefly summarizes our general approach. We refer to [5, 6] for details. For simplicity, we focus on sequential planning but with minor modifications the methods work for temporal planning as well. We assume the standard propositional STRIPS model of planning. A cost is associated to each action ($cost(a) > 0$), and the cost of a plan, which we seek to minimize, is the sum of the costs of the actions that are part of it. Our baseline planner, $\text{HSP}_0^*$, is based on regression and the $H^2$ heuristic, and uses IDA* to search. A search state, $s$, is a set of atoms, representing goals. Search starts from the set of goals ($G$) and ends when a set of goals satisfied by the initial state ($I$) is reached. The optimal cost of $s$, $H^*(s)$, is given by the Bellman equation [2]:

$$H^*(s) = \begin{cases} 0 & \text{if } s \subseteq I \\ \min_{s' \in R(s)} H^*(s') + \delta(s, s') & \end{cases}, \qquad (1)$$

where $R(s)$ is the set of states that can be constructed from $s$ by regression, and $\delta(s, s')$ is the cost of the action used to regress from $s$ to $s'$. The $H^m$ heuristic, obtained from (1) by introducing an admissible approximation, is defined by

$$H^m(s) = \begin{cases} 0 & \text{if } s \subseteq I \\ \min_{s' \in R(s)} H^m(s) + \delta(s, s') & \text{if } |s| \leqslant m \\ \max_{s' \subseteq s, |s'| \leqslant m} H^m(s') & \end{cases} \qquad (2)$$

For small $m$ (in practice, $m \leqslant 2$), $H^m(s)$ for all sets with $|s| \leqslant m$ can be efficiently computed by different dynamic programming or shortest path algorithms. The heuristic is stored in a table, and during search the cost of a state $s$ is given by the maximal cost of any subset of $s$ that is stored in the table[2]. This implies that as soon as a cost value for an atom set $s'$ is stored in the table, it becomes included in the subsequent evaluation of any set $s$ such that $s' \subseteq s$. We will use $T(s)$ to denote the estimated cost of $s$ given by the current table $T$.

## 3 Improving Heuristics through Search

For small values of $m$, the $H^m$ heuristic frequently underestimates the cost of both sets of more than $m$ atoms (because it considers only the most costly $m$-subset) and sets of $m$ or fewer atoms (because it does so recursively).

### 3.1 Boosting

Each set of atoms stored in the cost table (*table entry*) is itself a state in the search space (although perhaps not in the part of the search space explored by a search starting from the goal set, $G$). Thus, for any table entry $s$ we can find the optimal cost of $s$ by performing a search starting in $s$. Even if the search is interrupted before a solution is found, it may still result in an improved lower bound on the cost of $s$. We call this *boosting* the heuristic. To put this idea into practice, a number of questions have to be settled:

*Which table entries to boost?* Entries whose cost is already known to be optimal or infinite (*i.e.* unreachable) can obviously be ignored. Entries with a cost greater than $T(G)$ are also unlikely to be relevant, although this is not certain since the cost of $G$ may be underestimated.

*In what order?* Since entries with a smaller cost may be relevant in the boosting search for higher cost entries, but not vice versa, entries are boosted in order of increasing estimated cost.

*When to interrupt boosting searches?* To search until the optimal cost for each table entry has been found is generally too expensive to be practical. Two alternatives are: (*1*) Interrupt the search for an entry $s$ when it has been proven that the cost of $s$ is greater than the cost of the next entry in the list of entries to boost, thus preserving the "boost in order of increasing cost" principle. The boosting of an entry may be resumed once it has again become the least costly entry on the list. (*2*) Interrupt search after a fixed time (or some other measure of search effort)[3].

*When to stop the boosting process?* As entries are solved, proven unreachable or reach some search effort limit without cost improvement, the list of entries to boost will shorten and eventually become empty, at which point the boosting process comes to a natural end. A reasonable earlier stopping point is when the estimated cost of the

[1] Linköpings Universitet (pahas@ida.liu.se)

[2] The table is implemented as a Trie tree [1] so that enumerating all the subsets of $s$ that are not in the table can be avoided.

[3] When both conditions are applied, entries for which the search is interrupted without any improvement in cost are not reinserted into the list of entries to boost.

next (*i.e.* least costly) entry in the list is greater than $T(G)$, since the remaining entries are less likely to be relevant.

*Conflict Detection.* Boosting improves the cost estimates stored in the table. To improve estimates of cost for larger atoms sets, such sets have to be added to the table and boosted. However, adding entries indiscriminately is not cost-effective. As long as a set $s$ is not solved (*i.e.* its optimal cost not known), there is no reason to consider supersets of $s$ since further improving the cost estimate for $s$ also improves on all its supersets. Only when a set $s$ becomes solved do we consider adding new sets $s \cup \{p\}$, for atoms $p \notin s$. As an additional selection criterion, $s \cup \{p\}$ is added only if the plan found for $s$ deletes $p$[4]. A fixed limit on the size of any set added to the table can also be imposed, and this often improves performance.

## 3.2 Approximate Regression

The Bellman equation directly reflects the search space. Analogously, the equation that defines $H^m$ reflects a different search space, which we may call the $m$-*approximate* regression space. This is an AND/OR graph: Sets of $m$ or fewer atoms are OR-nodes, expanded by regression, while sets of more than $m$ atoms are AND-nodes, expanded by solving each subset of size $m$. The cost of an OR-node is minimized over all its children, while the cost of an AND-node is maximized.

To search in the approximate regression space, we use a modified IDA* algorithm, which we call IDAO* [5]. It differs from IDA* only in the DFS subroutine:

```
IDAO_DFS(s, b) {
 if final(s) {
  solved = true;
  return 0;
 }
 if (|s| > m) {  // AND-node
  for (each subset s' of s, |s'| == m) {
   c[s'] = IDAO(s', b);  // cost limit = b
   if (c[s'] > b) return c[s'];
  }
  solved = (all subsets solved);
  return max c[s'] over all s';
 } else {  // OR-node
  for (each s' in R(s)) {
   if (delta(s,s') + H(s') <= b) {
    c[s'] = delta(s,s') +
             IDAO_DFS(s', b - delta(s,s'));
    if (solved) return c[s'];
   } else {
    c[s'] = delta(s,s') + H(s');
   }
  }
  store H(s) = min c[s'] over all s', if improved;
  return min c[s'] over all s';
 }
}
```

IDAO_DFS discovers a lower bound on the cost of every encountered node (the optimal cost, if the node is solved). By storing the values of OR-nodes (size $m$ subsets) in the cost table, the search computes part of the $H^m$ heuristic as a side effect[6].

[4] The criterion can be strengthened or weakened by examining all optimal plans for $s$, but this is in general to expensive to be worthwhile. Several other selection criteria are also possible.

[5] IDAO* is not directly usable as a search algorithm for AND/OR graphs, because although it finds an optimal solution, it does not keep enough information for this solution to be extracted. It works for our purposes since we only need to know the optimal *cost*.

[6] A greater part of $H^m$ can be computed by searching all the children of each AND-node, but this is expensive and the value of the additional heuristic information is small.

IDAO* frequently encounters the same state (set of goals) more than once during search, and therefore it can be sped up (significantly) by storing solved nodes (both AND and OR) and short-cutting the search when it reaches a node that has already been solved. Unlike the lower bounds stored in the cost table, which are valid also for greater-$m$-approximate searches and in the original search space, the information in the "solved table" is valid only for the current $m$-approximate search.

## 4 Results, Conclusions and Related Work

Boosting and approximate regression can be incorporated into our basic planning algorithm in many ways. We have experimented with four variants: $\text{HSP}^*_\text{a}$ and $\text{HSP}^*_\text{b}$ use approximate search and boosting, respectively, to improve the heuristic before searching for a plan, while $\text{HSP}^*_\text{c}$ and $\text{HSP}^*_\text{d}$ interleave different heuristic improvement efforts with iterations of the main search. All four use $H^2$ as the base heuristic. Results indicate that the methods are cost-effective for some "combinatorially hard" problems, *i.e.* problems that are hard because of their structure and not only because of sheer size.

The idea of using search to derive or improve heuristics is not new: Pattern databases [3, 4] are constructed by search in a relaxed problem space, and solution cost in this space is used as a lower bound on cost in the original search space. Approximate regression can be viewed as such a relaxation. Pattern searches [7] discover the cost of "parts" of a search state, which are then used as lower bounds on the cost of any state containing those parts, similar to boosting.

The IDAO* algorithm is very similar (though not identical) to an iterative application of the Test subroutine of the SCOUT minimax search algorithm [8]. Although the idea of using Test in this iterative fashion, or enhancing it with memory, has, to our knowledge, not been applied before, there is a large body of work on *depth*-bounded (as opposed to *cost*-bounded) minimax search, *e.g.* [9], which could be used in an iterative deepening scheme instead of IDAO*. An alternative to our current use of approximate regression is to use a (depth-bounded) approximate search to improve the estimated cost of each state encountered in the main search.

## Acknowledgements

Héctor Geffner deserves as much credit for this work as I do (all faults are mine though).

## References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1987.

[2] R.E. Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[3] J.C. Culberson and J. Schaeffer, 'Searching with pattern databases', in *Canadian Conference on AI*, (1996).

[4] S. Edelkamp, 'Planning with pattern databases', in *Proc. 6th European Conference on Planning*, (2001).

[5] P. Haslum and H. Geffner, 'Admissible heuristics for optimal planning', in *Proc. 5th International Conference on Artificial Intelligence Planning and Scheduling*. AAAI Press, (2000).

[6] P. Haslum and H. Geffner, 'Heuristic planning with time and resources', in *Proc. 6th European Conference on Planning (ECP'01)*, pp. 121 – 132, (2001).

[7] A. Junghanns and J. Schaeffer, 'Sokoban: Enhancing general single-agent search methods using domain knowledge', *Artificial Intelligence*, **129**(1-2), 219 – 251, (2001).

[8] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, 1984.

[9] A. Plaat, J. Schaeffer, W. Pijls, and de Bruin A., 'Best-first fixed-depth minimax algorithms', *Artificial Intelligence*, **87**(1-2), 255 – 293, (1996).