Additive and Reversed Relaxed Reachability Heuristics Revisited

P@trik Haslum
NICTA & The Australian National University
patrik.haslum@anu.edu.au

Introduction & Motivation

HSP$^*_0$ and HSP$^*_F$ are sequential planners, optimal w.r.t. the sum of arbitrary (non-negative) action costs objective. Both are essentially STRIPS planners, though some additional language features, *e.g.*, the object fluents introduced in PDDL3.1, are handled by compilation. Both planners are based on heuristic state-space search, using the A* search algorithm with an admissible additive combination of h^2 heuristics, obtained via cost-distribution.

So, in what way do they differ? In the direction of the search: HSP$^*_0$ uses regression, searching backwards from the goal, while HSP$^*_F$ searches forwards from the initial state.

So, if that is the only difference, why enter both in the competition? There is experimental evidence that some planning domains are inherently more suited to forward search and others to regression, but this phenomenon is often obscured by other differences between the methods, such as, *e.g.*, heuristics, employed by planners. By creating forwards- and backwards-searching planners that are as similar as they can reasonably be, we may hope the competition results will provide a classification of the domains used w.r.t. this property.

Background

As usual, a propositional STRIPS planning problem (P) consists of a set of propositional symbols ("atoms"), a set of actions, an initial world state (I), and a set of goal atoms (G). Each action has a set of precondition atoms ($\mathrm{pre}(a)$) and sets of atoms made true ($\mathrm{add}(a)$) and false ($\mathrm{del}(a)$) by the action.

For reasons that will become clear, we assume that problems have a complete set of complementary atoms. That is, there is a total function, `not`, on the set of atoms such that $\texttt{not}(\texttt{not}(p)) = p$ for each p, and such that in any reachable state, exactly one of the atoms p and $\texttt{not}(p)$ is true. Complementary atoms are normally created when compiling away negations, but if required can be added to ensure completeness. This no more than doubles the number of atoms in the problem.

Each action a also has a (constant and independent) cost, $\mathrm{cost}(a)$, and the cost of a plan, $a_1, \ldots, a_n$, is the sum of the costs of actions in it: $\sum_{i=1,\ldots,n} \mathrm{cost}(a_i)$. Action costs are assumed to be non-negative, *i.e.*, $\mathrm{cost}(a) \geq 0$ for all a. In principle, costs may be arbtrary real numbers, but in practice, they are represented by rationals. (It is not obvious how, or even if, non-rational constants can be specified in PDDL.)

The (sequential) h^m heuristic function assigns to any set of atoms s a lower bound on the cost of any plan for achieving s, *i.e.*, reaching a state in which all atoms in s are true, from the initial state. It is defined by

$$h^m(s) = \begin{cases} 0 & \text{if } s \subseteq I \\ \min\limits_{\{a \mid \mathrm{del}(a) \cap s = \emptyset\}} h^m((s - \mathrm{add}(a)) \cup \mathrm{pre}(a)) + \mathrm{cost}(a) & \text{if } |s| \leqslant m \\ \max\limits_{s' \subseteq s, |s'| \leqslant m} h^m(s') & \text{else.} \end{cases}$$

Note that the atom set on the right-hand side in the second case is the *regression* of s through the action a, and that the minimisation is over all actions that are applicable to s in a regression search. The solution to the h^m equation – for a given initial state – can be represented by a table containing h^m values for any set s of at most m atoms. Computing this table is polynomial in the number of atoms but exponential in m. The heuristic value of any atom set s can then be computed by evaluating only the last case of the equation, using the table to obtain values of all size m subsets of s. (For further details on h^m, see Haslum, 2006).

On the Compilation of Object Fluents

The main novelty in PDDL3.1 [1] is the introduction of *object fluents*, functions that take their value among problem objects. They can be used to encode multi-valued (finite-domain) state variables. Arguments of predicates and (object- or numeric-valued) functions in PDDL3.1 can be general terms, composed of object fluents, variables and constants.

Compilation of a problem with object fluents to common, "predicate-only", PDDL form is a straightforward application of the principle of the Skolem normal form

[1] `http://ipc.informatik.uni-freiburg.de/PddlExtension`

in first-order logic – that a function term corresponds to an existentially quantified variable – in reverse. Each n:ary object-valued function `(f ?a1 .. ?an)` is replaced by an $(n+1)$:ary predicate `(val-f ?a1 .. ?an ?v)`, and action effects arranged so that `(val-f ?a1 .. ?an ?v)` is true exactly when `(f ?a1 .. ?an) = ?v`. Each occurrance of an object function term is replaced by a new existentially quantified variable, constrained to have the value of the function by conjoining an instance of the corresponding predicate. For example, the condition `(= (f (g ?x)) (f ?y))` becomes

```
(exists (?v1 ?v2 ?v3)
  (and (val-g ?x ?v1) (val-f ?v1 ?v2)
       (val-f ?y ?v3) (= ?v2 ?v3)))
```

Note that in a "STRIPS-style" action, existentially quantified variables can be transformed into additional parameters, thus preserving action simplicity.

There are (at least) two ways to represent that `(f ..)` is undefined: either no instance of `(val-f ..)` is true, or `(val-f .. U)` is true for a distinct constant "`U`". Both are workable, but the latter has some advantages: one is that exactly (instead of at most) one instance of `(val-f .. ?v)` is true in every reachable state, and hence that universal quantification over `?v` is equivalent to existential quantification. This greatly simplifies the compilation of conditional effects.

Planning Problem Reversal

The h^m heuristic is inherently geared to regression search: it estimates the cost of reaching any subgoal set (*i.e.*, regression search state) from a given world state. So, how can it be applied in the forward search carried out by HSP$^*_{\text{F}}$? By reversing the problem.

For every STRIPS planning problem P, there is a problem $R(P)$, constructible from P mechanically in polynomial time and space, such that for any valid plan $a_1, \ldots, a_n$ for $R(P)$ the reversed action sequence, *i.e.*, $a_n, \ldots, a_1$, is a valid plan for P, and vice versa. In fact, the correspondance between P and $R(P)$ goes deeper: for any (completely specified) state S and any plan achieving the goal G from S in P, the reversed action sequence is a plan achieving the set of atoms that hold in S from the initial state in $R(P)$, and therefore any lower bound on the cost of achieving S in $R(P)$ is also a lower bound on the cost of reaching the goal from S in P. Thus, problem reversal gives a basis for using the h^m heuristic in a forward search. The definition of STRIPS problem reversal and the "correspondance property" is due to Massey (1999), who used it to investigate directional bias in different planners. Pettersson (2005) used it to construct a reversed planning graph, thus creating a forward-searching version of Graphplan.

Massey's reversal procedure is compositional, creating for each action a in the original problem P a reversed action $R(a)$. Because the existence of a complementary atom, $\texttt{not}(p)$, for each atom p in P is assumed, the set of atoms in $R(P)$ will also be the same as in P. However, in $R(P)$ it may be possible to reach states in which both p and $\texttt{not}(p)$ are true, at least for some p. Intuitively, the preconditions of $R(a)$ are the effects of a, *i.e.*, what must in the original problem hold in any state immediately after a has been applied, and the effects of $R(a)$ "undo" the effects of a, so that in a state resulting from application of $R(a)$ in the reversed problem, the preconditions of a hold. Formally,

$$
\begin{aligned}
&\text{pre}(R(a)) = \\
&\quad \text{add}(a) \cup \{\texttt{not}(p) \mid p \in \text{del}(a)\} \cup (\text{pre}(a) - \text{del}(a)) \\
&\text{add}(R(a)) = \\
&\quad (\text{pre}(a) \cap \text{del}(a)) \cup \\
&\quad \{p, \texttt{not}(p) \mid p \in (\text{add}(a) \cup \text{del}(a)) \wedge \texttt{not}(p) \notin \text{pre}(a)\} \\
&\text{del}(R(a)) = \\
&\quad \{\texttt{not}(p) \mid p \in \text{pre}(a)\} \qquad .
\end{aligned}
$$

For any atom p mentioned in the (positive or negative) effects of a such that neither p nor $\texttt{not}(p)$ is in the precondition of a, $R(a)$ makes both p and $\texttt{not}(p)$ true, because in this situation it can not be determined if p was true or false before a was applied. The initial state of $R(P)$ assigns p true and $\texttt{not}(p)$ false for any atom $p \in G$, because any such atom must be true in a goal-satsifying state in the original problem, and true to both p and $\texttt{not}(p)$ for any atom not mentioned in G, because the status of such atoms in the goal state is unknown. The goal of $R(P)$ comprises all atoms that are true in the initial state of P (since each atom has a complement, taking only the atoms true in a state as goals in the reversed problem suffices to specify a world state completely). As noted above, this extends to any world state in P.

As noted above, it is in $R(P)$ possible to reach states in which both p and $\texttt{not}(p)$ are true, for some atoms p (indeed, this is usually true even in the initial state of $R(P)$). That there is in spite of this a one-to-one correspondance between plans for $R(P)$ and plans for P is due to the fact that the goal of $R(P)$ specifies a complete state. However, it typically makes heuristics computed on the reversed problem less informative – at least that is the case h^2. This problem can, at least to some extent, be countered by extending action preconditions and the goal in the original problem with relevant *implied* atoms before reversal. In many planning problems, certain properties are invariant over all reachable states. Of particular interest here are binary mutex relations: pairs of atoms p and q such that at most one is true in any reachable state. If atom q is mutex with some atom $p \in \text{pre}(a)$, $\texttt{not}(q)$ can be added to $\text{pre}(a)$ without changing the semantics of the problem (because in any reachable state where a is applicable, p is true, so q must be false and hence $\texttt{not}(q)$ true). This is done, if possible, for any atom that is, or whose complement is, mentioned by the actions effects. The goal G is likewise extended. Mutex information is obtained from the h^2 heuristic, for the original problem P: if $h^2(\{p, q\}) = \infty$, p and q are mutex.

Note that this reasoning can be extended also to "exactly-1" invariants, *i.e.*, sets of atoms of which exactly one is true in every reachable state: if $\{p_1, \ldots, p_k\}$

is such an invariant, and if every p_i in this set except one is mutex with atom set s, the single atom that is not must be true in any state where s holds, *i.e.*, is implied by s.

Additivity via Cost-Distribution

The h^m heuristic approximates the cost of achieving any set of more than m atoms by the cost of achieving the most costly subset of size m. In planning with sum of action costs as the objective, this sometimes results in poor estimates. This is most easily seen in problem with a large ($\gg m$) number of goals and where the sets actions relevant to achieving each goal have little overlap. In such cases, being able to add heuristic estimates of the cost of achieving separate sets of subgoals, without loss of admissibility, is crucial for the success of heuristic search.

Cost-distribution is a general method for obtaining an additive version of any admissible heuristic. The idea is simple: Given a planning problem P, create k "copies" of it, and modify the cost function in each copy so that for no action is the sum of its cost over all the copies greater than its cost in P. Then, the sum of optimal solution costs over all copies is also no greater than the optimal solution cost in P, and thus the sum of values of any ensamble of admissible heuristics, one computed on each copy, is admissible for P. This idea is implicit in some work on pattern database heuristics in domain-specific search (*e.g.*, Korf & Felner, 2002), and was generalised when applied to the h^2 heuristic by Haslum, Bonet & Geffner (2005). They, however, considered only "0–1" distribution of action costs, *i.e.*, counting the full cost of each action in one problem copy only and assigning it zero cost in all other copies. Katz & Domshlak (2007) recognised that the principle applies equally to any splitting of the cost of each action into fractions among the copies, as long as they do not sum to more than the original action cost.

The accuracy of an additive heuristic obtained via cost-distribution typically depends very much on the specific distribution of costs. A poor choice may even result in a heuristic that is weaker than the heuristic computed on the problem with original costs. Finding a good distribution of costs in a domain-independent manner is a non-trivial problem. Haslum, Bonet & Geffner (2005) proposed one strategy for partitioning actions (*i.e.*, constructing a 0–1 distribution), which creates one partition for each goal atom, by dividing actions into disjoint sets, based on the multi-valued state variable they affect, and assigning each set to one partition based on an estimate of the loss in heuristic value for the corrsponding goal atom incurred by not counting the cost of actions in the set. While this method achieves good results in some domains (notably Blocksworld), it also has several drawbacks: due to the initial division into action sets, it depends on a "good" choice of multi-valued state variable encoding, and creating one partition per goal atom is only appropriate when the solution divides into mostly independent parts at that level. As an example, in the Logistics and Rovers domains, a partition that groups actions by the "location" that they move something to (or otherwise affect) can result in a far more accurate additive h^2 heuristic. This partition can not be found by the above method.

At present, the method of cost distribution to be used in HSP^*_0 and HSP^*_F is still undecided, and several methods are under development. The following is only a sketch of some ideas currently pursued.

Analysing the Critical Tree Although the equation that defines the h^m heuristic is normally explained as a relaxation of the optimal cost function for the (sequential) regression search space, it may also be viewed as defining the optimal cost function for a different, relaxed, search space. This is a min/max (AND/OR) space, in which sets of more than m atoms are max/AND nodes, whose solution cost is the maximum over all child nodes, which are subsets of size m, and sets of m or fewer atoms are min/OR nodes, whose solution cost is the minimum over all child nodes, which are the possible regressions of the set. A max/AND node is solved only when all its children are solved, while a min/OR node is solved if one child node is solved. Thus, the solution, for a given goal set of atoms s, in this space is a tree. Call the subtree of this tree containing only branches corresponding to the most expensive children of each max node (*i.e.*, the ones that define its value) the *critical tree* of s.

Not all actions are represented in the critical tree. However, this does not mean that the cost of any action that is not can be ignored, because doing so may lower the cost some child of a min node below the value of the previously minimum-cost child, thus altering the shape of the tree. A *critical action set*, again for a given goal set of atoms s, is one such that counting the (full) cost of all actions in it is sufficient to preserve the cost of each node in the critical tree of s: to do so, it is sufficient to preserve for every max node the cost of at least one maximum-cost child node, and for every min node that the cost of all child nodes remains above (or equal to) the minimum-cost child node. This reasoning can be generalised to preserving any cost bound, yielding the following, non-deterministic, algorithm for computing a set $\text{CAS}(s, c)$ of actions, counting the cost of which is sufficient to ensure that the h^m cost of atom set s is at leact c:

If $c = 0$: $\text{CAS}(s, c) = \emptyset$.

If $|s| > m$, $c > 0$: **choose** $s' \subset s$ with $|s'| = m$ and $h^m(s') \geq c$, and let $\text{CAS}(a, s) = \text{CAS}(a, s')$; if there is no such s', **fail**.

If $|s| \leq m$, $c > 0$: Let $A = \emptyset$.

For each action a such that $\text{del}(a) \cap s = \emptyset$:

Let $s' = (s - add(a)) \cup \text{pre}(a)$; and

choose if $h^m(s') + \text{cost}(a) \geq c$, let $A = A \cup \{a\} \cup \text{CAS}(s', c - \text{cost}(a))$, else **fail**;

or choose if $h^m(s') \geq c$, let $A = A \cup \mathrm{CAS}(s', c)$, else **fail**.

Let $\mathrm{CAS}(s, c) = A$.

This yields a simple, recursive strategy for creating a 0–1 distribution (*i.e.*, a partioning of the set of actions): First, compute h^2, counting the cost of all actions, and find a (minimal) critical action set for the goal, G. They form the first partition. Next, compute h^2 counting the cost only of remaining actions and find a (minimal) critical action set for the goal with respect to this heuristic. This is the second partition. The process can repeated as many times as desired, until no actions remain.

The advantage of this strategy is that it preserves at least the h^m value of the goal set G. It also allows the maximum number of partitions created to be controlled. The main difficulty is efficiently realising the above algorithm (even without minimality).

Bottom-Up Construction (with Glueing) This strategy proceeds in a manner quite the opposite of those discussed above. Instead of slicing the set of actions in a top-down fashion along the top-level goals or the critical tree, it creates one initial action set for each atom, containing those actions that add the atom, and one problem copy for each action set. As the sets are not necessarily disjoint, the cost of an action represented in more than one set is split, evenly, between the relevant problem copies. This simple operation in some cases creates quite good additive heuristics.

However, it also has drawbacks: First, it leads to a very large number of h^2 components in the additive heuristic, making it costly to evaluate. Second, if the (most costly) preconditions of actions with the same effect are disjoint, the heuristic value can drastically decrease. To see why, suppose there are two actions, a_1 and a_2, that add p, and that $\mathrm{pre}(a_1) = \{q\}$ and $\mathrm{pre}(a_2) = \{r\}$. In the component counting actions that add p, the heuristic value of the preconditions of both a_1 and a_2 is zero (since no action relevant to achieving them counts), so the h^2 estimate of the cost of achieving p is just the minimum of $\mathrm{cost}(a_1)$ and $\mathrm{cost}(a_2)$. In the component counting actions that add q, the estimated cost of $\mathrm{pre}(a_1)$ is non-zero, but $h^2(\mathrm{pre}(a_2))$ is zero (since actions relevant to achieving r do not count) and thus $h^2(\{p\}) \leq h^2(\mathrm{pre}(a_2)) + \mathrm{cost}_q(a_2) = 0$ (where cost_q is the cost function in the problem copy counting only actions that add q, which is zero for a_2), and the same happens with a_1 in the component for r. Thus, even the sum over all three yields only $\min(\mathrm{cost}(a_1), \mathrm{cost}(a_2))$ for the cost of achieving p, ignoring the cost of achieving $\mathrm{pre}(a_1)$ and $\mathrm{pre}(a_2)$. In short, excessive splitting creates (alternative) "short-cuts" for achieving the same atom set in different copies, and because each component heuristic locally choses the cheapest way to achieve a set of atoms, the sum becomes low.

A way to counter both problems is to form a smaller number of larger sets of atoms (and placing in the corresponding action set each action that adds some atom in the set, again splitting the cost of actions relevant to more than one). In line with the bottom-up approach, atom sets are obtained by starting from a collection of singleton sets for each atom and iteratively merging (or "glueing") them. Some reasonable guidelines for the choice of sets to merge are:

1. Merge atoms that appear as (most costly) preconditions of different actions with overlapping effects. This to counter the problem illustrated by the example above.

2. Avoid merging atoms that appear together in action preconditions, or in the goal. Since all atoms in the precondition of an action, or in the problem goal, need to be achieved simultaneously, if at all, they do not give rise to the problem of "alternative short-cuts" illustrated above. Thus, adding rather than just maximising over them is likely to be better.

3. If atom p is relevant only as a precondition of some action that adds a (relevant) atom q, merge p with q. This should have little effect on the heuristic value, as the two atoms are steps along a single (critical) path rather than on alternative paths, but still serves to reduce the number of distinct sets.

4. Prefer merging sets of atoms whose sets of relevant actions have a large overlap, as this reduces the splitting of action costs into fractions.

These guidelines are not precise, probably not complete, and clearly sometimes contradictory. Exactly how to design a merging process based on them, or some other principle, is still an open question.

Acknowledgement NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proc. 20th National Conference on AI (AAAI'05)*, 1163–1168.

Haslum, P. 2006. *Admissible Heuristics for Automated Planning*. Ph.D. Dissertation, Linköpings Universitet.

Katz, M., and Domshlak, C. 2007. Structural patterns of tractable sequentially-optimal planning. In *Proc. of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, 200–2007.

Korf, R., and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134(1-2):9–22.

Massey, B. 1999. *Directions in Planning: Understanding the Flow of Time in Planning*. Ph.D. Dissertation, University of Oregon.

Pettersson, M. 2005. Reversed planning graphs for relevance heuristics in AI planning. In Castillo, L.; Borrajo, D.; Salido, M.; and Oddi, A., eds., *Planning, Scheduling and Constraint Satisfaction: From Theory to Practice*, volume 117 of *Frontiers in AI and Applications*. IOS Press. 29–38.