Towards Rational Deployment of Multiple Heuristics in A* (Extended Abstract)

David Tolpin, Tal Beja
Solomon Eyal Shimony
CS Department
Ben-Gurion University
Israel
tolpin,bejat,shimony}@cs.bgu.ac.il

Ariel Felner
ISE Department
Ben-Gurion University
Israel
felner@bgu.ac.il

Erez Karpas
IEM Department
Technion
Israel
karpase@gmail.com

Abstract
In this paper we discuss and experiment with Lazy A*, a variant of A* where heuristics are evaluated lazily and with rational lazy A*, which decides whether to compute the more expensive heuristics at all, based on a myopic value of information estimate. Full version appears in IJCAI-2013 [Tolpin et al., 2013]

1 Lazy A*
This paper examines the case where we have several available admissible heuristics. Clearly, we can evaluate all these heuristics, and use their maximum as an admissible heuristic, a scheme we call A_{MAX}. The problem with naive maximization is that all the heuristics are computed for all the generated nodes. In order to reduce the time spent on heuristic computations, Lazy A* (or LA*, for short) evaluates the heuristics one at a time, lazily. When a node n is generated, LA* only computes one heuristic, h_1(n), and adds n to OPEN. Only when n re-emerges as the top of OPEN is another heuristic, h_2(n), evaluated; if this results in an increased heuristic estimate, n is re-inserted into OPEN. This idea was briefly mentioned by Zhang and Bacchus (2012) in the context of the MAXSAT heuristic for planning domains. LA* is as informative as A_{MAX}, but can significantly reduce search time, as we will not need to compute h_2 for many nodes. In this paper we provide a deeper examination of LA* and describe several technical optimizations for LA*.

The pseudo-code for LA* is depicted as Algorithm 1, and is very similar to A*. In fact, without lines 7–10, LA* would be identical to A* using the h_1 heuristic. When a node n is generated we only compute h_1(n) and n is added to OPEN (Lines 11–13), without computing h_2(n) yet. When n is first removed from OPEN (Lines 7–10), we compute h_2(n) and reinsert it into OPEN, this time with f_{MAX}(n).

It is easy to see that LA* is as informative as A_{MAX}, as they both generate and expand the same set of nodes (up to differences caused by tie-breaking). The reason is that a node n is expanded by both A_{MAX} and by LA* when f_{MAX}(n) is the best f-value in OPEN.

In its general form A* generates many nodes that it does not expand. These nodes, called surplus nodes [Felner et al., 2012], are in OPEN when we expand the goal node with f = C*. LA* avoids h_2 computations for many of these surplus nodes. By contrast, A_{MAX} computes both h_1 and h_2 for all generated nodes. Thus, LA* can potentially run faster than A_{MAX} in many cases.

2 Rational Lazy A*
LA* offers us a very strong guarantee, of expanding the same set of nodes as A_{MAX}. However, often we would prefer to expand more states, if it means reducing search time. We now present Rational Lazy A* (RLA*), an algorithm which attempts to optimally manage this tradeoff.

Using principles of rational meta-reasoning [Russell and Wefald, 1991], theoretically every algorithm action (heuristic function evaluation, node expansion, open list operation) should be treated as an action in a sequential decision-making meta-level problem: actions should be chosen so as to achieve the minimal expected search time. However, the appropriate general meta-reasoning problem is extremely hard to define precisely and to solve optimally.

Therefore, we focus on just one decision type, made in the context of LA*, when n re-emerges from OPEN (Line 7). We have two options: (1) Evaluate the second heuristic h_2(n) and add the node back to OPEN (Lines 7–10) like LA*, or (2) bypass the computation of h_2(n) and expand n right way.
The only addition of $RLA^*$ to $LA^*$ is the option to bypass $h_2$ computations (Lines 7-10). Suppose that we choose to compute $h_2$ — this results in one of the following outcomes: 1: $n$ is still expanded, either now or eventually.

2: $n$ is re-inserted into OPEN, and the goal is found without ever expanding $n$.

Computing $h_2$ is helpful only in outcome 2, where potential time savings are due to pruning a search subtree at the expense of $t_2(n)$. Since we do not know this in advance, we calculate and use $p_h$ — the probability that $h_2$ is helpful.

In order to choose rationally, we define a criterion based on value of information (VOI) of evaluating $h_2(n)$ in this context. The following notations are used. $b(n)$ is the branching factor at node $n$, $t_2$ is the time to compute $h_2$ and re-insert $n$ into OPEN thus delaying the expansion of $n$. $t_1$ is the time to remove $n$ from OPEN and $p_l$ the probability that $h_2$ is helpful.

As we wish to minimize the expected regret, we should thus evaluate $h_2$ just when:

$$(1 - b(n)p_h)t_d < p_l t_e$$

And bypass this computation otherwise. The complete derivation appears in our full paper [Tolpin et al., 2013].

3 Experimental results

We experimented with $LA^*$ and $RLA^*$ on a number of domains but focus here on planning domains where we experimented with two state of the art heuristics: the admissible landmarks heuristic $h_{LMCUT}$ [Helmert and Domshlak, 2009], and the landmark cut heuristic $h_{LMCUT}$ [Helmert and Domshlak, 2009] (used as $h_2$). We experimented with all planning domains without conditional effects and derived predicates (which the heuristics we used do not support) from previous IPCs.

Table 1 depicts the experimental results (for two of our domains and the overall over all domains) for $LA^*$ and $RLA^*$ to that of $A^*$ using each of the heuristics individually, as well as to their max-based combination, and their combination using selective max (Selmax) [Domshlak et al., 2012]. Selmax is an online learning scheme which chooses one heuristic to compute at each state. The leftmost part of the table shows the number of solved problems in each domain. As the table demonstrates, $RLA^*$ solves the most problems, and $LA^*$ solves the same number of problems as selective max. Thus, both $LA^*$ and $RLA^*$ are state-of-the-art in cost-optimal planning.

The middle part of the Table 1 shows the geometric mean of planning time in each domain, over the commonly solved problems (i.e., those that were solved by all 6 methods). $RLA^*$ is the fastest overall, with $LA^*$ second. Of particular interest is the miconic domain. Here, $h_{LMCUT}$ is very informative and thus the variant that only computed $h_{LA}$ is the best choice (but a bad choice overall). Observe that both $LA^*$ and $RLA^*$ saved 86% of $h_{LMCUT}$ computations, and were very close to the best algorithm in this extreme case. This demonstrates their robustness.

The rightmost part of Table 1 shows the average fraction of problems for which $LA^*$ and $RLA^*$ did not evaluate the more expensive heuristic, $h_{LMCUT}$, over the problems solved by both these methods. This is shown in the good columns. We can see that in domains where there is a difference in this number between $LA^*$ and $RLA^*$, $RLA^*$ usually performs better in terms of time. This indicates that when $RLA^*$ decides to skip the computation of the expensive heuristic, it is usually the right decision.

Finally, Table 2 shows the total number of expanded and generated states over all commonly solved problems. $LA^*$ is indeed as informative as $A^*_{MAX}$ (the small difference is caused by tie-breaking), while $RLA^*$ is a little less informed and expands slightly more nodes. However, $RLA^*$ is much more informative than its “intelligent” competitor - selective max, as these are the only two algorithms in our set which selectively omit some heuristic computations. $RLA^*$ generated almost half of the nodes compared to selective max, suggesting that its decisions are better.

Table 1: Planning Domains — Number of Problems Solved, Total Planning Time, and Fraction of Good Nodes

<table>
<thead>
<tr>
<th>Domain</th>
<th>$h_{LA}$</th>
<th>lmcut</th>
<th>max</th>
<th>selmax</th>
<th>$LA^*$</th>
<th>$RLA^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problems Solved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>miconic</td>
<td>141</td>
<td>140</td>
<td>140</td>
<td>141</td>
<td>141</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>722</td>
<td>722</td>
<td>722</td>
<td>722</td>
<td>722</td>
<td>0.57</td>
</tr>
<tr>
<td></td>
<td>23797219</td>
<td>23797219</td>
<td>23797219</td>
<td>23797219</td>
<td>23797219</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>108201244</td>
<td>108201244</td>
<td>108201244</td>
<td>108201244</td>
<td>108201244</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>0.87</td>
<td>0.87</td>
<td>0.87</td>
<td>0.87</td>
<td>0.87</td>
<td>0.88</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>27</td>
<td>27</td>
<td>27</td>
<td>27</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>197972262</td>
<td>197972262</td>
<td>197972262</td>
<td>197972262</td>
<td>197972262</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>110839548</td>
<td>110839548</td>
<td>110839548</td>
<td>110839548</td>
<td>110839548</td>
<td>0.95</td>
</tr>
</tbody>
</table>

Table 2: Total Number of Expanded and Generated States

<table>
<thead>
<tr>
<th>Domain</th>
<th>Expanded</th>
<th>Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_{LA}$</td>
<td>183320267</td>
<td>1184443684</td>
</tr>
<tr>
<td>lmcut</td>
<td>23797219</td>
<td>114315382</td>
</tr>
<tr>
<td>$A_{MAX}$</td>
<td>22774804</td>
<td>108132460</td>
</tr>
<tr>
<td>selmax</td>
<td>54557689</td>
<td>193980693</td>
</tr>
<tr>
<td>$LA^*$</td>
<td>22790804</td>
<td>108207244</td>
</tr>
<tr>
<td>$RLA^*$</td>
<td>25742262</td>
<td>110935698</td>
</tr>
</tbody>
</table>

References


