ECAI 2014
T. Schaub et al. (Eds.)

doi:10.3233/978-1-61499-419-0-1107

Rational Deployment of Multiple Heuristics in IDA*

David Tolpin [1] and **Oded Betzalel** [2] and **Ariel Felner** [3] and **Solomon Eyal Shimony**[4]

1 Introduction

In the past, we adapted metareasoning techniques to decide on whether to evaluate a costly heuristic in A* [6]. Since IDA* [3] is a linear-space simulation of A*, similar techniques are likely to speed up IDA* as well - the focus of this paper. The first thing to consider is lazy IDA*: lazy evaluation of the heuristics. Given heuristics h_1 and h_2, if h_1 causes a cutoff there is no need to evaluate h_2.

The main contribution of this paper is *Rational lazy IDA** (RL-IDA*) which uses a myopic expected regret criterion whether to skip evaluation of h_2 after computation of h_1 *fails* to cut off search. We provide experimental results on sliding tile puzzles and on the container relocation problem [7], showing that RLIDA* outperforms both IDA* and LIDA*.

2 Lazy IDA*

Assume for clarity only two available admissible heuristics, h_1 and h_2, that h_1 is faster to compute than h_2, but h_2 is *weakly more informed*, i.e., $h_1(n) \leq h_2(n)$ for most of the nodes n. We denote the cost of the optimal solution by C^*, and computation time of h_1 and of h_2 by t_1 and t_2, respectively. Unless stated otherwise we assume that t_2 is much greater than t_1.

Let T be the current IDA* threshold. After $h(n)$ is evaluated, if $g(n) + h(n) > T$, then n is pruned and IDA* backtracks to n's parent. Given both h_1 and h_2, a naive implementation of IDA* will evaluate them both and use their maximum in comparing against T. In the context of IDA*, if $g(n)+h_1(n) > T$ the search can backtrack without the need to compute h_2, resulting in Lazy IDA* (depicted in Algorithm 1). The "optional condition" in line 15 is needed for the *rational* lazy A* algorithm, described below: in lazy IDA*, this condition is always true.

Algorithm 1: Lazy IDA*

```
1  Lazy-IDA* (root) {
2      Let Thresh = max(h1(root), h2(root))
3      Let solution = null
4      while solution == null and Thresh < ∞ do
5          solution, Thresh = Lazy-DFS(root, Thresh)
6      return solution
7  }
8  Lazy-DFS(n, Thresh) {
9      if g(n) > Thresh then
10         return null, g(n)
11     if goal-test(n) then
12         return n, Thresh
13     if g(n)+h1(n) > Thresh then
14         return null, g(n)+h1(n)
15     if opt-cond and g(n)+h2(n) > Thresh then
16         return null, g(n)+h2(n)
17     Let next-Thresh = ∞
18     for n' in successors(n) do
19         Let solution, temp-Thresh = Lazy-DFS-lim(n', Thresh)
20         if solution ¬ = null then
21             return solution, temp-Thresh
22         else
23             Let next-Thresh = min(temp-Thresh, next-Thresh)
24     return null, next-Thresh
25 }
```

3 Rational Lazy IDA*

Meta-reasoning [5] is a general theory, hard to apply in practice, except under specific assumptions and simplifications. We focus on deciding whether to evaluate or to bypass the computation of h_2 in the context of IDA*. Each IDA* iteration is a depth-first search up to a gradually increasing threshold T. For each node n, we say that evaluating $h(n)$ is *helpful* if $g(n) + h(n) > T$, i.e. the heuristic *helped* in the sense that node n is pruned in this iteration.

The only addition of Rational Lazy IDA* to Lazy IDA* is the option to bypass $h_2(n)$ computations (line 15). Suppose that we choose to compute h_2 — this results in one of the following outcomes:

1. $h_2(n)$ is not helpful and n is expanded.
2. $h_2(n)$ is helpful (because $g(n) + h_2(n) > T$), pruning n, which is not expanded in the current IDA* iteration.

Computing h_2 can be *beneficial* only in outcome 2, but the outcome is known to the algorithm only *after* h_2 is computed. The decision on whether to evaluate h_2 must be based on the subjective probability of each of the outcomes. The time wasted by being suboptimal in deciding whether to evaluate h_2 is called the *regret* of the decision. We make the following meta-level assumptions:

1. The decision is made *myopically*: assumes that the algorithm continues like Lazy IDA* starting with the children of n.
2. h_2 is *consistent*: if evaluating h_2 is beneficial on n, it is also beneficial on any successor of n.
3. h_1 will not cause pruning in any of the children of n.

Table 1 summarizes the regret of each possible decision, for each possible future outcome. t_e is time to expand n, and $b(n)$ the number of its children. Denote the probability that $h_2(n)$ is helpful by p_h.

[1] CS Department, Ben-Gurion University. E-mail:tolpin@cs.bgu.ac.il
[2] CS Department, Ben-Gurion University. E-mail:odedbetz@cs.bgu.ac.il
[3] ISE Department, Ben-Gurion University. E-mail:felner@bgu.ac.il
[4] CS Department, Ben-Gurion University. E-mail:shimony@cs.bgu.ac.il

	Compute h_2	Bypass h_2
h_2 helpful	0	$t_e + b(n)t_1 + (b(n)-1)t_2$
h_2 not helpful	t_2	0

Table 1: Time losses in Rational Lazy IDA*

The expected regret of computing $h_2(n)$ is thus $(1 - p_h)t_2$. On other hand, the expected regret of bypassing $h_2(n)$ is $p_h(t_e + b(n)t_1 + (b(n)-1)t_2)$. As we wish to minimize the expected regret, we should thus evaluate h_2 just when:

$$(1 - p_h b(n))t_2 < p_h(t_e + b(n)t_1) \quad (1)$$

If $p_h b(n) \geq 1$ (the left side of Equation 1 is negative), then the expected regret is minimized by always evaluating h_2, regardless of the values of t_1, t_2 and t_e. For $p_h b(n) < 1$, the decision of whether to evaluate h_2 depends on the values of t_1, t_2 and t_e:

$$\textbf{evaluate } h_2 \textbf{ if } t_2 < \frac{p_h}{1 - p_h b(n)}(t_e + b(n)t_1) \quad (2)$$

The factor $\frac{p_h}{1-p_h b(n)}$ depends on the potentially unknown probability p_h, making it difficult to reach the optimum decision. However, if our goal is just to do better than Lazy IDA*, then it is "safe" to replace p_h by an upper bound on p_h. Assume (as a first approximation) that values of h_1 and h_2 are iid. Denote: $x = 1 - \frac{h_1(n)}{\max(h_1(s), h_2(n))}$. Also, let $\overline{x_N}$ denote the average of N samples of x, and define: $l = 1 - \frac{h_1(n)}{T - g(n)}$. Then using the union bound and the Hoeffding and Markov inequalities, we can (nontrivially) obtain the bound:

$$p_h \leq \frac{1 + \sqrt{\log \sqrt{2N}l}}{\sqrt{2N}l} + \frac{\overline{x_N}}{l} \quad (3)$$

4 Evaluation on sliding tile puzzles

For consistency of comparison, we used for the 15 puzzle 98 out of Korf's 100 tests [3]: all tests solved using under 20 minutes with standard IDA* with h_1 being Manhattan Distance (MD). As h_2 we used the *linear-conflict heuristic* (LC) [4].

algorithm	time	generated	h_2 total	h_2 helpful
IDA* (MD)	58.84	268,163,969		
IDA* (LC)	40.08	30,185,881		
LIDA*	32.85	30,185,881	21,886,093	6,561,972
RLIDA*	20.09	47,783,019	8,106,832	4,413,050
Clairvoyant	12.66	30,185,881	6,561,972	6,561,972

Table 2: Results for 15 puzzle

Results (Table 2) are for a constant $p_h = 0.3$, estimated from trial runs of RLIDA* on a few problem instances. The advantage of Rational Lazy IDA* is evident: even though it expands many more nodes than Lazy IDA*, its runtime is significantly lower as it saves even more time on evaluations of LC. The *Clairvoyant* row is an unrealizable oracle scheme that evaluates h_2 only if helpful.

algorithm	time	generated	h_2 total	h_2 helpful
IDA* (MD)	184.46	822,898,188		
IDA* (LC)	155.35	104,943,867		
LIDA*	112.74	104,943,890	65,660,207	12,549,104
RLIDA*	63.08	137,881,842	21,564,188	8,871,727
Clairvoyant	40.36	104,943,890	12,549,104	12,549,104

Table 3: Results for weighted 15 puzzle

Table 3 shows results for 82 of Korf's 100 initial positions on weighted (move cost equals number on the tile) 15 puzzle solved in under 20 minutes by IDA*. Rational Lazy A* achieves a significant speedup here as well. Similar results occured in 3*5 and 3*6 puzzles.

5 Evaluation on container relocation problem

The container relocation problem is encountered in retrieving stacked containers for loading onto a ship in sea-ports [7]. We are given S stacks of containers, each stack with up to T containers. The initial state has $N \leq S \times T$ containers, arbitrarily numbered from 1 to N. Rules of stacking containers are as in blocks world. The goal is to "retrieve" all containers in order of number, from 1 to N, and place them on a freight truck that takes them away. The objective function to minimize is the number of container moves until all containers are gone. We assume the "restricted" version [7].

Every container numbered X which is above at least one container numbered $Y < X$ must be relocated. The number of such containers is used as h_1 (called LB_1 in [7]). Counting one more for each container that must be relocated a second time as any place to which it is moved will block some other container, is used as h_2 (LB_3 in [7]).

algorithm	time	generated	h_2 total	h_2 helpful
IDA* (LB_1)	372	853,094,579		
IDA* (LB_3)	704	110,753,768		
LIDA*	368	130,695,270	42,862,888	19,060,111
RLIDA*, $p_h = 0.3$	337	233,077,220	27,628,566	13,575,017
RLIDA*, $p_h \leq 0.5$	320	158,362,305	33,693,072	16,460,400
Clairvoyant	194	130,695,270	19,060,111	19,060,111

Table 4: Results for container relocation

Results are shown in Table 4 for the 49 hardest tests out of those solved in under 20 minutes using LB_1, from the CVS test suite described in [1, 2]. Rational Lazy IDA* improves performance even when p_h was assumed constant ($P_h = 0.3$). As the branching factor is almost constant (frequently equal to the number of stacks minus 1), there is room for improvement by better estimating p_h. Using the bounds developed in Section 3 to estimate p_h dynamically indeed achieves this (line RLIDA*, $p_h \leq 0.5$).

6 Acknowledgments

Supported by the Israeli Science Foundation (ISF) grant #417/13, and by the Lynne and William Frankel center for computer science.

REFERENCES

[1] Marco Caserta, Stefan Voβ, and Moshe Sniedovich, 'Applying the corridor method to a blocks relocation problem', *OR Spectr.*, **33**(4), 915–929, (October 2011).
[2] Bo Jin, Andrew Lim, and Wenbin Zhu, 'A greedy look-ahead heuristic for the container relocation problem', in *IEA/AIE*, volume 7906 of *LNCS*, pp. 181–190. Springer, (2013).
[3] R. E. Korf, 'Depth-first iterative-deepening: An optimal admissible tree search', *Artificial Intelligence*, **27**(1), 97–109, (1985).
[4] Richard E. Korf and Larry A. Taylor, 'Finding optimal solutions to the twenty-four puzzle', in *AAAI*, pp. 1202–1207, (1996).
[5] Stuart Russell and Eric Wefald, 'Principles of metereasoning', *Artificial Intelligence*, **49**, 361–395, (1991).
[6] D. Tolpin, T. Beja, S. E. Shimony, A. Felner, and E. Karpas, 'Toward rational deployment of multiple heuristics in A*', in *IJCAI*, (2013).
[7] Huidong Zhang, Songshan Guo, Wenbin Zhu, Andrew Lim, and Brenda Cheang, 'An investigation of IDA* algorithms for the container relocation problem', in *Proc. of the 23rd Inter. Conf. on Industrial Engineering and Other Applications of Applied Int. Systems - Part I*, IEA/AIE'10, pp. 31–40, Berlin, Heidelberg, (2010). Springer-Verlag.