Type System Based Rational Lazy IDA*

Oded Betzalel
Computer Science Dept.
Ben Gurion University
Beer-Sheva, Israel 84105
odedbetz@cs.bgu.ac.il

Ariel Felner
Information Systems Engineering Dept.
Ben Gurion University
Beer-Sheva, Israel 84105
felner@bgu.ac.il

Solomon Eyal Shimony
Computer Science Dept.
Ben Gurion University
Beer-Sheva, Israel 84105
shimony@cs.bgu.ac.il

Abstract

Meta-reasoning can improve numerous search algorithms, but necessitates collection of statistics to be used as probability distributions, and involves restrictive meta-reasoning assumptions. The recently suggested scheme of type systems in search algorithms is used in this paper for collecting these statistics. The statistics are then used to better estimate the unknown quantity of expected regret of computing a heuristic in Rational Lazy IDA* (RLIDA*), and also facilitate a second improvement due to relaxing one of the unrealistic meta-reasoning assumptions in RLIDA*.

Introduction

All search algorithms have decision points on how to continue search. Traditionally, tailored rules are hard-coded into the algorithms. Meta-reasoning techniques based on value of information or other ideas can significantly speed up the search, by making these rules dependent on expected benefit (runtime and/or result quality). This was shown for depth-first search in CSPs (Tolpin and Shimony 2011), for Monte-Carlo tree search (Hay et al. 2012), and recently for A* (Tolpin et al. 2013) and IDA* (Tolpin et al. 2014).

As stated in (Tolpin et al. 2013; Hay et al. 2012; Tolpin et al. 2014), success of meta-reasoning depends heavily on the type of meta-reasoning assumptions made in developing the meta-reasoning rules, and on estimating prior distributions over outcomes of evaluating heuristics. A typical case we examine here is Rational Lazy IDA* (RLIDA*), a state of the art variant of IDA* appearing in (Tolpin et al. 2014). In RLIDA*, two heuristics, $h_1$ and $h_2$, are available, and meta-reasoning is used to make the RLIDA* decision: whether to compute $h_2(n)$ after having seen the value $h_1(n)$ for a search node $n$. The decision is based on an expected regret criterion, computed under a set of meta-reasoning assumptions. This paper re-examines the meta-reasoning assumptions used in RLIDA*, and introduces two improvements: a “relaxed” RLIDA*, where one unrealistic meta-reasoning assumption of RLIDA* is relaxed, and a better scheme to estimate the needed distributions.

The latter estimation issue is handled through the notion of type systems, recently coined in (Lelis, Zilles, and Holte 2013), although it was used before (Korf, Reid, and Edelkamp 2001). A type system partitions the state-space nodes into “similar valued” classes of nodes, called types. Type systems were typically used to predict the number of nodes generated by search algorithms. In this paper we use the type system attributes as features in collecting statistics. These statistics are then used in the RLIDA* decision.

In (Tolpin et al. 2014), results on RLIDA* were compared to to an unrealizable “clairvoyant” scheme that computes $h_2$ only if it is helpful. RLIDA* was close in preformance to the clairvoyant scheme for tile puzzles, and we thus did not expect to get additional improvement here. But for the container relocation problem (Zhang et al. 2010), RLIDA* did much worse that “clairvoyant”, so we expected to do better using the methods proposed here. These expectations were indeed confirmed by the experimental results.

Rational Lazy IDA*

We begin by briefly revisiting lazy IDA* and its rational version, RLIDA* (Tolpin et al. 2014). We are given two admissible heuristics, $h_1$ and $h_2$, which take time $t_1$ and $t_2$ to compute, respectively. The assumption is that $h_2$ is more informed than $h_1$, but $t_2$ is much greater than $t_1$, so the main question is whether one should invest the time to compute $h_2$ after obtaining the value of $h_1$ at a node $n$.

Let $T$ be the current IDA* threshold. After $h(n)$ is evaluated, if $f(n) = g(n) + h(n) > T$, 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 Lazy IDA* and its rational variant, RLIDA*, the idea is to avoid computing the heavy $h_2$ as much as possible, which can sometimes cost additional node expansions.

The pseudo-code for lazy IDA* and RLIDA* is depicted as Algorithm 1. If the trivial cutoff conditions in lines 8 or 10 do not hold, $h_1$ is evaluated first (line 12), and if $f_1$ is already above the threshold (i.e. $h_1$ causes a cutoff), the search (in both LIDA* and RLIDA*) backtracks (line 14). $h_2$ is thus only evaluated (in lines 15-18) if $f_1(n) ≤ T$. In lazy IDA*, the “optional condition” in line 15 is defined to be “always true”, so $h_2$ is always evaluated if $h_1$ failed to cause a cutoff. RLIDA* is more conservative in deciding to compute $h_2$, and the “optional condition” in line 15 is based on an estimated regret criterion (Tolpin et al. 2014), i.e. how much...
Algorithm 1: Rational 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 = Lazy-DFS(root, Thresh)
6.     return solution
7. }

Lazy-DFS(n, Thresh) {
8.     if g(n) > Thresh then
9.         return null, g(n)
10.    if goal-test(n) then
11.       return n, Thresh
12.    Compute h1 and update statistics
13.    if g(n)+h1(n) > Thresh then
14.       return null, g(n)+h1(n)
15.    if opt-cond then
16.       Compute h2 and update statistics
17.       if g(n)+h2(n) > Thresh then
18.           return null, g(n)+h2(n)
19.     Let next-Thresh = ∞
20.     for n’ in successors(n) do
21.         Let solution, temp-Thresh = Lazy-DFS(n’, Thresh)
22.         if solution == null then
23.             return solution, temp-Thresh
24.         else
25.             Let next-Thresh = min(temp-Thresh, next-Thresh)
26.     return null, next-Thresh
}

extra computation time is wasted vs. the post-facto correct decision. Note that, like IDA*, RLIDA* returns the optimal solution regardless of how opt-cond is defined and computed. The regret estimate in (Tolpin et al. 2014) is based on the following meta-reasoning assumptions:

1. The decision is made myopically: we work under the assumption that opt-cond will be true in every child n’ of n, and therefore that h2(n’) will always be computed if h1(n’) does not cut off n’.
2. h2 is consistent: if evaluating h2(n) causes a cutoff in n, it also causes a cutoff in every child of n.
3. h1 will not cause a cutoff in any of the children of n.

There are two possible outcomes for h2:

**Outcome 1**: h2(n) does not cause a cutoff. Thus, if we choose to bypass computation of h2(n), we lose nothing. But if we compute h2(n), the effort (time t2) invested in evaluating h2(n) is wasted.

**Outcome 2**: h2(n) does cause a cutoff (henceforth called “helpful”). So, if we compute h2(n) we lose nothing. But if we bypass h2(n), we needlessly expand n. Due to assumptions 1 and 3, we now evaluate both h1 and h2 for all children of n. Since h2(n) is helpful, the children of n will not be expanded in this IDA* iteration due to assumption 2.

Since we do not know which outcome is true before h2(n) is computed, we assume a known probability p_{h2}(n) for outcome 2, i.e. that h2(n) is helpful. Denoting the time to expand n by t_e(n), and the branching factor (number of children) of n by b(n), minimizing the expected regret defined by the above quantities results in the following expression as opt-cond (Tolpin et al. 2014):

\[ t_2 < \frac{p_{h2}(n)}{1-p_{h2}(n)b(n)}(t_e(n) + b(n)t_1) \]  

(1)

**RLIDA* issues, proposed improvements**

Relaxed RLIDA*

Assumption 3 in RLIDA*, that h1 will not cause a cutoff in any of the children of the current node n, is obviously frequently violated. We relax it so that there is some probability p_{h1}(n) that such pruning will occur. (We assume that p_{h1}(n) is i.i.d. for all the children of n.)

<table>
<thead>
<tr>
<th>Compute h2</th>
<th>Bypass h2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2 uninitialized</td>
<td>0</td>
</tr>
<tr>
<td>t2 initialized</td>
<td>t_e(n)+h1(n)(t_e(n)+b(n)(1-p_{h1}(n)))+t2-t2</td>
</tr>
</tbody>
</table>

Table 1: Regret in Relaxed Rational Lazy IDA*

Table 1 summarizes the regret of each possible decision and outcome. The regret values are as in RLIDA* (Tolpin et al. 2014), except the case: bypassing h2 when it is helpful. Here we lose t_e by expanding n, plus the need to evaluate h1 in each of the children of n, and also evaluate h2 in each child with probability 1 − p_{h1}(n). From this value we subtract t2 from the regret, as we saved the time to compute h2.

We thus wish to bypass h2(n) only when:

\[ (1-p_{h2})t_2 < p_{h2}(t_e + b(n)t_1 + (b(n)(1-p_{h1}) - 1)t_2) \]  

And so we get a new opt-cond:

\[ t_2 < \frac{p_{h2}(n)}{1-p_{h2}(n)b(n)(1-p_{h1}(n))}(t_e(n) + b(n)t_1) \]  

(2)

We denote the improved RLIDA* that uses Eq. 3 as opt-cond, by “relaxed” RLIDA*, or RRLIDA* for short.

**Type systems in RLIDA* and RRLIDA***

Despite the simplicity of equations 1,3, their implementation is non-trivial, because all of the quantities b(n), t1, t2, t_e(n), and especially p_{h1}(n) and p_{h2}(n) may actually be unknown. One can try to estimate these unknown values either off-line or on-the-fly. But the way these values change between different nodes varies quite drastically across different problem types, and this is especially true for the probability terms. For some domains, such as sliding tile puzzles, the algorithm speedups were only slightly affected by the estimate of p_{h2}(n) over a wide range of values of p_{h2}(n), and the determining factor is b(n) which is known. In general, 1 p_{h2}(n) is denoted p_{h}(n) in (Tolpin et al. 2014), but we added the additional subscript as we also have a p_{h1}(n) later on.
however, the estimate of $p_{h_2}(n)$ can be crucial, and better results can be achieved if $p_{h_2}(n)$ is a variable which depends on some features of $n$. In order to assess $p_{h_2}$ we tried the following type systems:

**Type System 1**: (TS1) TS1 is simply the value of $h_1$. A table counting occurrences of $h_2$ values as a function of $h_1$ is generated on the fly in “update statistics” (lines 12, 16 in Algorithm 1). These are used to estimate $p_{h_2}$, the probability that $h_2$ will be helpful for the current node. (Note that this information is available only for nodes where $h_2$ was evaluated.) When used in conjunction with RLIDA* or RRLIDA*, we use the type system name to indicate the improved algorithm variant, e.g. RLIDA*+TS1 is RLIDA* using type system TS1 for estimating $p_{h_2}$.

**Type System 2**: (TS2) TS2 ignores important information, by assuming that the distribution of $h_2$ given $h_1$ is constant in the entire search tree, which is unlikely to be true. TS2 creates table of counts similar to TS1, but also as a function the value of $h_2$ in the closest ancestor for which $h_2$ was computed, and of the distance to that ancestor.

To estimate $p_{h_2}(n)$ using a type system, we divide the number of times $h_2$ was greater than $T - g(n)$, by the total count for that row, in the counting table for the appropriate type. (Recall that $T$ is the current IDA*/RLIDA* threshold).

**Type systems for $p_{h_1}$**: we could not effectively assess $p_{h_1}$ without a type system. To assess $p_{h_1}(n)$ we need information on $h_1$ in $n$’s children, which is insensitive to obtain. Instead we use a type system that lists the distribution of $h_1(n)$ based on $h_1$ at the parent of $n$, as a proxy for estimating $p_{h_1}(n)$. We tried several type systems that yielded similar results, the results shown in this paper are for the type that counts $h_1(n)$ occurrences as a function of $h_1(\text{parent}(n))$, and the distance to the last $h_2$ computation. To estimate $p_{h_1}(n)$, divide the number of times where $h_1$ was greater than $T - g(n) - \text{cost}(n, n')$, by the total count for that row in the counting table. In container relocation and standard sliding tile puzzles, $\text{cost}(n, n') = 1$.

### Empirical evaluation

As this paper is an attempt to improve RLIDA*, we mostly use the same problems and problem instance sets as in (Tolpin et al. 2014).

### Sliding tile puzzles

We first examine the results on the 15-puzzle. For consistency of comparison, we used as test cases for the 15 puzzle the same 98 out of Korf’s 100 instances (Korf 1985) as in (Tolpin et al. 2014): all the tests that were solved in less than 20 minutes with standard IDA* using the Manhattan Distance (MD) heuristic. The $h_2$ heuristic was the linear-conflict heuristic (LC) (Korf and Taylor 1996) which adds a value of 2 to MD for pairs of tiles that are in the same row (or the same column) as their respective goals but in a reversed order. One of these tiles will need to move away from the row (or column) to let the other pass.

Results for this problem set are shown in Table 2, listing average runtime in seconds, number of generated nodes, number of $h_2$ evaluations, and number of helpful $h_2$ evaluations. As expected, RLIDA* with an assumed constant $p_{h_2} = 0.3$, outperforms all other algorithms, an exception being the unrealizable “Clairvoyant” algorithm, which evaluates $h_2$ only if it turned out to be helpful. (The results for this “algorithm” are obtained by not counting the runtime for the $h_2$ evaluations which did not cause a cutoff.)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Generated</th>
<th>$h_2$ Total</th>
<th>$h_2$ Helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDA* (MD)</td>
<td>58.84</td>
<td>268,163,969</td>
<td>6,561,972</td>
<td>6,561,972</td>
</tr>
<tr>
<td>IDA* (LC)</td>
<td>40.08</td>
<td>30,185,881</td>
<td>21,886,093</td>
<td>7,313,390</td>
</tr>
<tr>
<td>LIDA*</td>
<td>32.85</td>
<td>30,185,881</td>
<td>8,106,832</td>
<td>4,413,050</td>
</tr>
<tr>
<td>RLIDA*</td>
<td>20.09</td>
<td>47,783,019</td>
<td>8,106,832</td>
<td>4,413,050</td>
</tr>
<tr>
<td>Clairvoyant</td>
<td>12.66</td>
<td>30,185,881</td>
<td>6,561,972</td>
<td>6,561,972</td>
</tr>
<tr>
<td>RLIDA*+TS1</td>
<td>33.90</td>
<td>49,314,132</td>
<td>14,265,984</td>
<td>9,315,480</td>
</tr>
<tr>
<td>RLIDA*+TS2</td>
<td>27.49</td>
<td>39,466,460</td>
<td>11,508,429</td>
<td>7,313,390</td>
</tr>
<tr>
<td>RRLIDA*+TS1</td>
<td>35.08</td>
<td>65,289,319</td>
<td>7,908,331</td>
<td>5,351,080</td>
</tr>
<tr>
<td>RRLIDA*+TS2</td>
<td>30.23</td>
<td>57,518,574</td>
<td>6,719,180</td>
<td>4,625,750</td>
</tr>
</tbody>
</table>

Table 2: Results for 15 puzzle

### Container relocation problem

The container relocation problem is an abstraction of a planning problem encountered in retrieving stacked containers for loading onto a ship in sea-ports (Zhang et al. 2010). We are given $S$ stacks of containers, where each stack consists of up to $T$ containers. The initial state has $N \leq S \times T$ containers, arbitrarily numbered from 1 to $N$. The rules of stacking and of moving containers is the same as for blocks in the blocks-world domain. The goal is to “retrieve” all containers in order of number, from 1 to $N$, i.e. to place them on a freight truck that takes the container away to be loaded onto a ship. The objective function to minimize is the number of container moves until all containers are gone. The complication comes from the fact that we can only “retrieve” a container if it is at the top of one of the stacks. Optimally solving this problem is NP-hard (Zhang et al. 2010).

As in (Tolpin et al. 2014), we assume here the version where each container (“block” in blocks-world terminology) is uniquely numbered, that a stack $s$ that currently has $T$ containers is “full” and no additional containers can be placed on $s$ until some container is moved away from the top of $s$.

The heuristics we used as $h_1$ for the experiments are the
same as in (Tolpin et al. 2014), as summarized below. Every container numbered $X$ which is above at least one container $Y$ with a number smaller than $X$ must be moved from its stack in order to allow $Y$ to be retrieved. The number of such containers in a state is used as the admissible heuristic $h_1$ called $LB_1$ in (Zhang et al. 2010). $h_2$ adds one relocation for each container that must be relocated a second time as any place to which it is moved will block some other container. This heuristic is called $LB_2$ in (Zhang et al. 2010). In the experiments, we used the same 49 instances as in (Tolpin et al. 2014): the hardest instances out of those that were solved within 20 minutes with the $LB_1$ heuristic, from the CVS test suite described in (Caserta, Voβ, and Sniedovich 2011; Jin, Lim, and Zhu 2013). Mean results are shown in Table 3 (left). In this domain Rational Lazy IDA* shows some performance improvement over lazy IDA*. However, as noted in (Tolpin et al. 2014), there was much room for improvement by making $p_{h_2}$ dependent on features of the search node, achieved here by using type systems. Thus, we tried to estimate $p_{h_2}(n)$ differently for each node, as a function of the node type. We report results for type systems TS1 and TS2, for both RLIDA* and RRLIDA*. Indeed (see Table 3(left)), using type systems to estimate $p_{h_2}$ improves RLIDA*, but the most significant improvement here is due to dropping assumption 3, that $h_1$ is unhelpful in the children (RRLIDA*).

We then conjectured that the timing differences should increase when we include harder problem instances, and added an additional 14 instances with 5 stacks and 9 or 10 tiers, with a runtime greater than 20 minutes in IDA*, selected arbitrarily from the above mentioned test-set. The results (Table 3 (right)), are mean values, for both the smaller and larger instances. Using type systems with RLIDA* was faster than LIDA*, but was actually slower than just using IDA* with only $h_1$. The reason is evident from examining the line “IDA*(LB3)”. Although $h_2$ drastically reduces generated node numbers, its runtime with these larger problem instances outweighed its usefulness to such an extent that RLIDA* can at best approach IDA* by evaluating $h_2$ very rarely. But lifting the assumption that $h_1$ does not cause a cutoff in the children (RRLIDA*) achieves further speedup and the best performance of all. The difference between TS1 and TS2 does not appear significant: TS2 achieves better accuracy, but has higher overhead for meta-reasoning, so in overall runtime performance, TS2 is usually only slightly better than TS1, and sometimes slightly worse. (In large instances, we had a time-out of twice the time of “IDA*(LB1)”, hence some discrepancies in node counts).

Note that in both container relocation results RRLIDA* performs better than the clairvoyant scheme, which seems surprising. However, upon deeper examination, it turns out that even if $h_2$ cuts off a node $n$ after $h_1$ fails to do so, it does not follow that one should evaluate $h_2$. For example, consider a node $n$ with $b(n)$ children, where $h_2$ cuts off the search at node $n$, where $h_1$ would cut off the search at all of its $b(n)$ children. Then, if evaluating $h_2(n)$ is more expensive than computing $h_1$ for all $b(n)$ children, then bypassing $h_2$ may be better than evaluating it. RRLIDA* takes such cases into account, whereas the clairvoyant scheme does not.

### Conclusion

Rational Lazy IDA* (Tolpin et al. 2014) are an instance of the rational meta-reasoning framework (Russell and Wefald 1991). Applying this framework necessitates extreme meta-reasoning assumptions that are frequently violated in practice, and estimation of distributions. This paper improves upon RLIDA* in both aspects: it relaxes the meta-reasoning assumption that the “weak” $h_1$ heuristic causes no cutoff in the children of the current node. Estimating the distributions of the probability of pruning for both $h_1$ and $h_2$ is improved by leveraging the notion of type systems (Lelis, Zilles, and Holte 2013). Experimental results on container relocation and sliding tile puzzles show that these improvements of RLIDA* achieving better accuracy in deciding when $h_2$ is helpful. A overall runtime improvement occurs for container relocation, where there is significant room for improvement over RLIDA*, but not for sliding tile puzzles, where RLIDA* already did sufficiently well in this respect. It is also interesting that RRLIDA* sometimes outperformed the unrealizable “clairvoyant” scheme, due to relaxing a meta-reasoning assumption. While the latter improvement is specific to RLIDA*, applying type-systems to meta-reasoning can be generalized, especially to rational lazy A* (Tolpin et al. 2013) which faces a similar meta-level decision.

### Acknowledgments

This research was supported by the Israel Science Foundation (ISF) grant #417/13 and the Frankel Center For Computer Science.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Small instances</th>
<th>All instances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>generated</td>
</tr>
<tr>
<td>IDA* (LB1)</td>
<td>336</td>
<td>853,094,579</td>
</tr>
<tr>
<td>LIDA*</td>
<td>366</td>
<td>128,798,338</td>
</tr>
<tr>
<td>RLIDA* ($n_2 = 0.3$)</td>
<td>337</td>
<td>233,077,220</td>
</tr>
<tr>
<td>Clairvoyant</td>
<td>228</td>
<td>128,798,338</td>
</tr>
<tr>
<td>RRLIDA*+TS1</td>
<td>327</td>
<td>166,781,023</td>
</tr>
<tr>
<td>RRLIDA*+TS2</td>
<td>922</td>
<td>159,923,334</td>
</tr>
<tr>
<td>RRLIDA*+TS1</td>
<td>207</td>
<td>318,146,242</td>
</tr>
<tr>
<td>RRLIDA*+TS2</td>
<td>201</td>
<td>300,623,173</td>
</tr>
</tbody>
</table>

Table 3: Container Relocation: 49 small instance (left), all 63 instances(right).
References


