Towards Rational Deployment of Multiple Heuristics in A*

David Tolpin Tal Beja Solomon Eyal Shimony
Ariel Felner
Erez Karpas

ICAPS 2013 Workshop on Heuristic Search for Domain-Independent Planning

Outline

Motivation

- We want to find an optimal solution, and we have admissible heuristics
- Use A^*

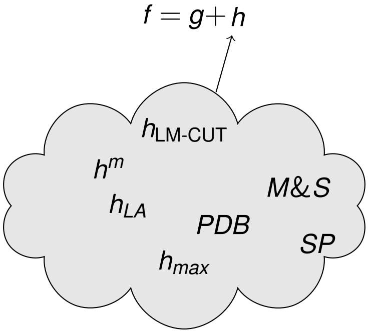
Motivation

- We want to find an optimal solution, and we have admissible heuristics
- Use A^*

$$f = g + h$$

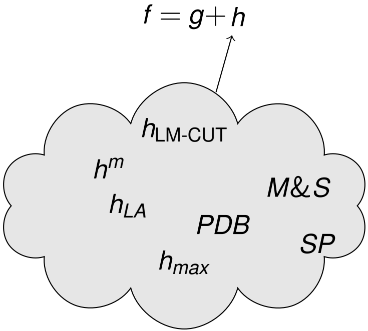
Motivation

- We want to find an optimal solution, and we have admissible heuristics
- Use A^*



Motivation

- We want to find an optimal solution, and we have admissible heuristics
- Use A^*



Which heuristic is the best?

Why Settle for One?

- There is no single best heuristic, so why settle only for one?
- We can use the maximum of several heuristics to get a more informative heuristic

Why Settle for One?

- There is no single best heuristic, so why settle only for one?
- We can use the maximum of several heuristics to get a more informative heuristic

Why Settle for One?

- There is no single best heuristic, so why settle only for one?
- We can use the maximum of several heuristics to get a more informative heuristic
- Sample results:

Domain	h_{LA}	$h_{\text{LM-CUT}}$	$\max_h$
openstacks-opt11	10	**11**	9
freecell	**54**	10	36

Number of problems solved in 5 minutes

Why Settle for One?

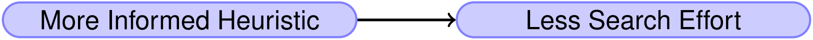
- There is no single best heuristic, so why settle only for one?
- We can use the maximum of several heuristics to get a more informative heuristic
- Sample results:

Domain	h_{LA}	$h_{\text{LM-CUT}}$	$\max_h$
openstacks-opt11	10	**11**	9
freecell	**54**	10	36

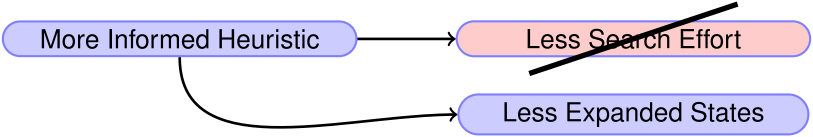
Number of problems solved in 5 minutes

- A more informed heuristic — $\max_h$— solves less problems

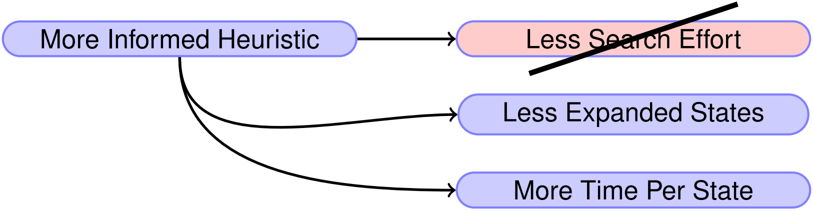
The Accuracy / Computation Time Tradeoff



The Accuracy / Computation Time Tradeoff



The Accuracy / Computation Time Tradeoff



Related Work

- Selective Max (Domshlak, Karpas and Markovitch 2012)
  - Uses a classifier which tries to predict which heuristic to use for each state
  - Classifier is learned online, during search
- Lazy A^* (Zhang and Bacchus, 2012)
  - Calculates heuristics only “as needed” to push a state further back in the open list

Contributions

- Theoretical analysis of lazy A^*
- Enhancements for lazy A^*
- Rational lazy A^* — applies rational meta-reasoning to decide whether or not to push a state back in the open list

Outline

Notation and Assumptions

- Two heuristics: h_1 and h_2
- h_1 is cheaper to compute than h_2
- h_2 is more informative than h_1 on average

- h_1 computation time is t_1, h_2 computation time is t_2
- Open list insertion/removal takes t_o time

A^*

Apply all heuristics to initial state s_0
Insert s_0 into OPEN
while OPEN *not empty* **do**
    $n \leftarrow$ best node from OPEN
    if *Goal(n)* **then**
        return trace(n)
    foreach *child c of n* **do**
        Apply h_1 to c
        insert c into OPEN
    Insert n into CLOSED
return FAILURE

Lazy A^*

```
Apply all heuristics to initial state s_0
Insert s_0 into OPEN
while OPEN not empty do
    n ← best node from OPEN
    if Goal(n) then
        return trace(n)
    if h_2 was not applied to n then
        Apply h_2 to n
        insert n into OPEN
        continue        //next node in OPEN
    foreach child c of n do
        Apply h_1 to c
        insert c into OPEN
    Insert n into CLOSED
return FAILURE
```

Analysis of Lazy A^*

- As informative as A^* using $\max(h_1, h_2)$ (up to tie-breaking)
- The *surplus* states are those that were generated but not expanded (i.e., on the open list) when A^*_{MAX} terminates
- Out of the surplus states, lazy A^* skips h_2 computation for some — denote them as *good*

	Expanded	Surplus	
Alg		Non good	Good
A^*_{MAX}	$t_1 + \mathbf{t_2} + 2t_o$	$t_1 + \mathbf{t_2} + t_o$	$t_1 + \mathbf{t_2} + t_o$
LA^*	$t_1 + \mathbf{t_2} + 4t_o$	$t_1 + \mathbf{t_2} + 3t_o$	$t_1 + t_o$

- If $g(s) + h_1(s) > C^*$ then s will be good (if it is generated)

Enhancements for Lazy A^*

- Open bypassing
  - When state s is generated and $h_1(s)$ computed, if $f(s)$ is smaller than the lowest f-value on *OPEN*, compute $h_2(s)$ right away
  - When computing $h_2(s)$, if the new $f(s)$ is smaller than the lowest f-value on *OPEN*, expand s right away
  - Reduces the overhead on *OPEN* operations
- Heuristic bypassing
  - Suppose we can derive upper and lower bounds for $h_1(s)$ and $h_2(s)$, e.g., when the heuristics are consistent
  - With lazy A^*, if we can prove that $\overline{h_1(s)} < \underline{h_2(s)}$, we use $\underline{h_2(s)}$ instead of computing h_1
  - We can also skip computing $h_2(s)$ when $\overline{h_2(s)} \leq h_1(s)$

Outline

Rational Lazy A^*

- Sometimes, it's better to expand more states in less time
- Lazy A^* does not consider this option
- We introduce *Rational Lazy* A^*, which differs from lazy A^* by deciding whether or not to compute h_2
- The decision is based on rational meta-reasoning

Rational Decision

- When should we decide to compute h_2?
- Assume we computed h_2 for state s. Then either:
  1. s will be expanded later
  2. s will not be expanded before the goal is found
- We should only compute h_2 if outcome 2 will occur — call this h_2 being helpful

Rational Decision

- When should we decide to compute h_2?
- Assume we computed h_2 for state s. Then either:
  1. s will be expanded later
  2. s will not be expanded before the goal is found
- We should only compute h_2 if outcome 2 will occur — call this h_2 being helpful

Rational Decision

- When should we decide to compute h_2?
- Assume we computed h_2 for state s. Then either:
  1. s will be expanded later
  2. s will not be expanded before the goal is found
- We should only compute h_2 if outcome 2 will occur — call this h_2 being helpful

Rational Decision

- When should we decide to compute h_2?
- Assume we computed h_2 for state s. Then either:
  1. s will be expanded later
  2. s will not be expanded before the goal is found
- We should only compute h_2 if outcome 2 will occur — call this h_2 being helpful

> *"It is difficult to make predictions, especially about the future"*
>
> — *Yogi Berra / Neils Bohr*

Almost Rational Decision

- We look at an upper bound of the regret for each decision, under each possible future
- We assume rational lazy A^* is better than lazy A^*, so by assuming we continue with lazy A^* we get an upper bound on regret

	Compute h_2	Bypass h_2
h_2 helpful	0	$\sim b(s)t_1 + (b(s) - 1)t_2$
h_2 not helpful	$\sim t_2$	0

$b(s)$ denotes the number of successors of s

Disclaimer: for the exact analysis, see the paper

From Regret to Rational Decision

	Compute h_2	Bypass h_2
h_2 helpful	0	$\sim b(s)t_1 + (b(s)-1)t_2$
h_2 not helpful	$\sim t_2$	0

- Assume the probability of h_2 being helpful is p_h
- Then the rational decision is to compute h_2 iff:

$$\frac{t_2}{t_1} < \frac{p_h b(s)}{1 - p_h b(s)}$$

Approximating p_h

$$\frac{t_2}{t_1} < \frac{p_h b(s)}{1 - p_h b(s)}$$

- We can directly measure t_1, t_2 and $b(s)$, but we need to approximate p_h
- If s is a state at which h_2 was helpful, then we computed h_2 for s, but did not expand s. Denote the number of such states by B.
- Denote by A the number of states for which we computed h_2.
- We can use $\frac{A}{B}$ as an estimate for p_h
- To get an estimate which is more stable, we use a weighted average with k fictitious examples giving an estimate of p_{init}:

$$\frac{(A + p_{init} \cdot k)}{B + k}$$

- We use $p_{init} = 0.5$ and $k = 1000$

Outline

Planning Domains

Alg	Solved	623 Commonly Solved Time (GM)	Expanded	Generated
h_{LA}	698	1.18	183,320,267	1,184,443,684
$h_{\text{LM-CUT}}$	697	0.98	23,797,219	114,315,382
max	722	0.98	**22,774,804**	**108,132,460**
selmax	747	0.89	54,557,689	193,980,693
LA^*	747	0.79	**22,790,804**	**108,201,244**
RLA^*	**750**	**0.77**	25,742,262	110,935,698

- RLA^* solves the most problems, and is fastest on average
- LA^* is as informative as A^*_{MAX}
- Caveat: per individual domain, LA^*/ RLA^* are not always best

Weighted 15 Puzzle Experiments

- h_1 — weighted manhattan distance
- h_2 — lookahead to depth l with h_1
- Uses a different derivation for the rational decision rule, which does not ignore t_o

	Generated			Time		
l	A^*	LA^*	RLA^*	A^*	LA^*	RLA^*
2	1,206,535	1,206,535	1,309,574	0.707	0.820	0.842
4	1,066,851	1,066,851	1,169,020	0.634	0.667	0.650
6	889,847	889,847	944,750	**0.588**	0.533	0.464
8	740,464	740,464	793,126	0.648	**0.527**	0.377
10	611,975	611,975	889,220	0.843	0.671	**0.371**
12	**454,130**	**454,130**	807,846	0.927	0.769	0.429

Limitations of LA^*: 15 Puzzle Experiments

Alg.	Generated	HBP1	HBP2	OB	Bad	Good	time
$h1 = \Delta X$, $h2 = \Delta Y$, Depth = 26.66							
A*	1,085,156	0	0	0	0	0	**415**
A*+HBP	1,085,156	216,689	346,335	0	0	0	417
LA*	1,085,157	0	0	734,713	37,750	312,694	417
LA*+HBP	1,085,157	140,746	342,178	589,893	37,725	115,361	416
$h1$ = Manhattan distance, $h2$ = 7-8 PDB, Depth 52.52							
A*	43,741	0	0	0	0	0	34.7
A*+HBP	43,804	30,136	1,285	0	0	0	33.6
LA*	43,743	0	0	42,679	47	1,017	34.2
LA*+HBP	43,813	7,669	1,278	42,271	21	243	**33.3**

- The A^* / LA^* enhancements described above work “too well”
- The heuristics are relatively cheap compared to open list operations
- Thus there is little room for improvement by LA^*, while the overhead is significant

Summary

- LA^* is as informative as A^*_{MAX}, with less heuristic computation
- RLA^* applies rational meta-reasoning to LA^* and reduces search time

- RLA^* is much simpler to implement than selective max
- By making a decision when we already know that $g(s) + h_1(s) < C^*$, RLA^* can use a much simpler decision rule to greater benefit

Thank You