

# The Joy of Forgetting: Faster Anytime Search via Restarting

Silvia Richter

Griffith University & NICTA, Australia

Jordan T. Thayer & Wheeler Ruml

University of New Hampshire, US

July 10, 2009

# Origin: developing a planner for IPC-2008

IPC-2008 requirement: find best possible plan within 30 minutes.

This suggested an **anytime approach**:

- Find a solution as quickly as possible (any solution is better than none).  
~> greedy best-first search
- While there is still time, try to improve the solution.  
~> weighted  $A^*$  with decreasing weights

Interesting finding:

A series of **independent runs** of weighted  $A^*$  seemed to perform better than one **continued** search.

Basic algorithm:

- 1 Set **weight** and **bound**  
bound = cost of best known solution, initially  $\infty$
- 2 Update open list w. r. t. weight if necessary
- 3 Conduct WA\* search, using bound for pruning
- 4 Upon new best solution: report solution, goto 1.

Variants used in literature:

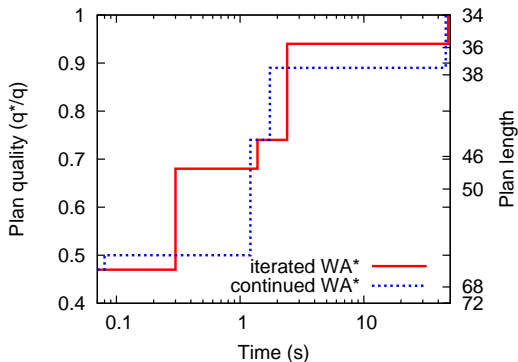
- Anytime A\* (Zhou & Hansen 2001, 2004)
- ARA\* (Likhachev et al. 2003)

# Example: Blocksworld task 11-2

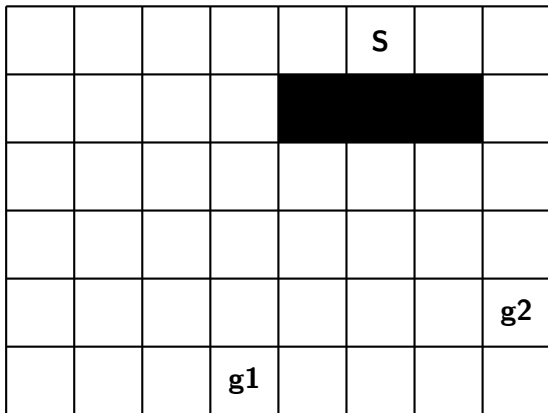
Plan lengths found over time:

- GBFS + iterated WA\*: 72 50 46 36 34
- GBFS + continued WA\*: 72 68 46 38 34

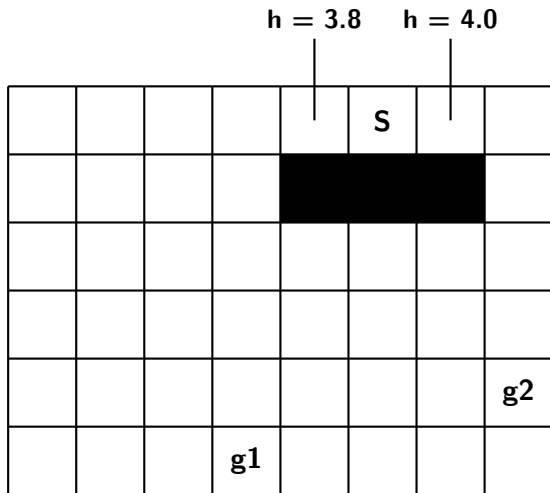
Plan qualities (best length / current length):



# The problem: low- $h$ bias

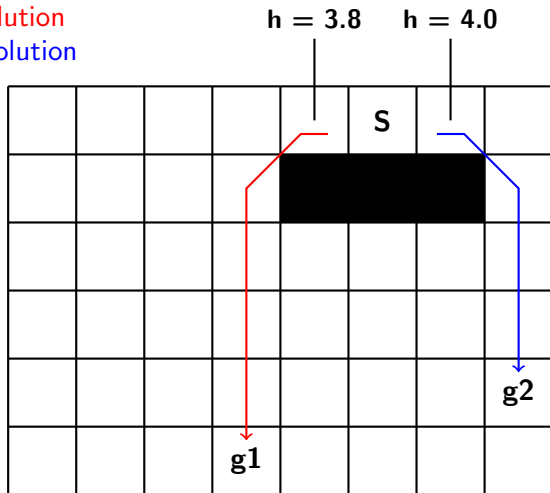


# The problem: low- $h$ bias



# The problem: low- $h$ bias

greedy solution  
optimal solution



# The problem: low- $h$ bias

h-values

less accurate the further from goal

less accurate on the left

		3.8	3.8	3.8	S	4.0	4.0
		3.4	3.4				3.0
	2.6	2.6	2.6	2.6	1.9	2.0	2.0
2.6	1.8	1.8	1.8	1.8	1.9	1.0	1.0
2.6	1.8	1.0	1.0	1.0	1.9	1.0	g2
	1.8	1.0	g1	1.0	1.9	1.0	1.0

# The problem: low- $h$ bias

$f'$ -values,  $w = 2$

		10.6	9.6	<u>8.6</u>	S	9.0	
		9.8	8.8				12.0
	9.2	8.2	8.2	8.2	7.8	9.0	10.0
10.2	7.6	7.6	7.6	7.6	7.8	7.0	8.0
10.2	8.6	7.0	7.0	7.0	8.8	7.0	g2
	9.6	8.0	g1	8.0	9.8	8.0	8.0

# The problem: low- $h$ bias

$f'$ -values,  $w = 2$

x expanded states

		10.6	9.6	8.6 x	S x	9.0	
		9.8	8.8 x				12.0
	9.2	8.2	8.2 x	8.2	7.8	9.0	10.0
10.2	7.6	7.6	7.6 x	7.6	7.8	7.0	8.0
10.2	8.6	7.0	7.0 x	7.0	8.8	7.0	g2
	9.6	8.0	g1 x	8.0	9.8	8.0	8.0

# The problem: low- $h$ bias

$f'$ -values,  $w = 2$

**x** expanded states

○ states in open list

		10.6	9.6	8.6 x	S x	9.0	
		9.8	8.8 x				12.0
	9.2	8.2	8.2 x	8.2	7.8	9.0	10.0
10.2	7.6	7.6	7.6 x	7.6	7.8	7.0	8.0
10.2	8.6	7.0	7.0 x	7.0	8.8	7.0	g2
	9.6	8.0	g1 x	8.0	9.8	8.0	8.0

# The problem: low- $h$ bias

$f'$ -values,  $w = 2$

**x** expanded states

○ states in open list

must expand for optimal path

		10.6	9.6	8.6 x	S x	9.0	
		9.8	8.8 x				12.0
	9.2	8.2	8.2 x	8.2	7.8	9.0	10.0
10.2	7.6	7.6	7.6 x	7.6	7.8	7.0	8.0
10.2	8.6	7.0	7.0 x	7.0	8.8	7.0	g2
	9.6	8.0	g1 x	8.0	9.8	8.0	8.0

# The problem: low- $h$ bias

$f'$ -values,  $w = 2$

must expand for optimal path

but many open states have lower  $f'$ -value

		10.6	9.6	8.6 x	S x	9.0	
		9.8	8.8 x	[Blackout]			12.0
	9.2	8.2	8.2 x	8.2	7.8	9.0	10.0
10.2	7.6	7.6	7.6 x	7.6	7.8	7.0	8.0
10.2	8.6	7.0	7.0 x	7.0	8.8	7.0	g2
	9.6	8.0	g1 x	8.0	9.8	8.0	8.0

# The problem: low- $h$ bias

$f'$ -values,  $w = 1.5$  (reduced weight)

$\rightsquigarrow$  search less greedy

		8.7	7.7	6.7 x	S x	7.0	
		8.1	7.1 x	[Blackout]			10.5
	7.9	6.9	6.9 x	6.9	6.85	8.0	9.0
8.9	6.7	6.7	6.7 x	6.7	6.85	6.5	7.5
8.9	7.7	6.5	6.5 x	6.5	7.85	6.5	g2
	8.7	7.5	g1 x	7.5	8.85	7.5	7.5

# The problem: low- $h$ bias

$f'$ -values,  $w = 1.5$  (reduced weight)

$\rightsquigarrow$  search less greedy

but effect still persists

		8.7	7.7	6.7 x	S x	7.0	
		8.1	7.1 x	[Blackout]			10.5
	7.9	6.9	6.9 x	6.9	6.85	8.0	9.0
8.9	6.7	6.7	6.7 x	6.7	6.85	6.5	7.5
8.9	7.7	6.5	6.5 x	6.5	7.85	6.5	g2
	8.7	7.5	g1 x	7.5	8.85	7.5	7.5

# The problem: low- $h$ bias

$f'$ -values,  $w = 1.5$  (reduced weight)

		8.7	7.7	6.7 x	S x	7.0	
		8.1	7.1 x	[REDACTED]			10.5
	7.9	6.9 x	6.9 x	6.9 x	6.85	8.0	9.0
8.9	6.7	6.7 x	6.7 x	6.7 x	6.85	6.5	7.5
8.9	7.7	6.5 x	6.5 x	6.5 x	7.85	6.5	g2
	8.7	7.5	g1 x	7.5	8.85	7.5	7.5

# The problem: low- $h$ bias

$f'$ -values,  $w = 1.5$  (reduced weight)

		8.7	7.7	6.7 x	S x	7.0	
		8.1	7.1 x	[REDACTED]			10.5
	7.9	6.9 x	6.9 x	6.9 x	6.85	8.0	9.0
8.9	6.7	6.7 x	6.7 x	6.7 x	6.85	6.5	7.5
8.9	7.7	6.5 x	6.5 x	6.5 x	7.85	6.5	g2
	8.7	7.5	g1 x	7.5	8.85	7.5	7.5

# The problem: low- $h$ bias

$f'$ -values,  $w = 1.5$  (reduced weight)

		8.7	7.7	6.7 x	S x	7.0	
		8.1	7.1 x	[Blackout]			10.5
	7.9	6.9 x	6.9 x	6.9 x	6.85 x	8.0	9.0
8.9	6.7 x	6.7 x	6.7 x	6.7 x	6.85 x	6.5	7.5
8.9	7.7	6.5 x	6.5 x	6.5 x	7.85	6.5	g2
	8.7	7.5	g1 x	7.5	8.85	7.5	7.5

# The problem: low- $h$ bias

$f'$ -values,  $w = 1.5$  (reduced weight)

		8.7	7.7	6.7 x	S x	7.0	
		8.1	7.1 x	[REDACTED]			10.5
	7.9	6.9 x	6.9 x	6.9 x	6.85 x	8.0	9.0
8.9	6.7 x	6.7 x	6.7 x	6.7 x	6.85 x	6.5	7.5
8.9	7.7	6.5 x	6.5 x	6.5 x	7.85	6.5	g2
	8.7	7.5	g1 x	7.5	8.85	7.5	7.5

# The problem: low- $h$ bias

$f'$ -values,  $w = 1.5$  (reduced weight)

		8.7	7.7	6.7 x	S x	7.0	
		8.1	7.1 x	[Blackout]			10.5
	7.9	6.9 x	6.9 x	6.9 x	6.85 x	8.0	9.0
8.9	6.7 x	6.7 x	6.7 x	6.7 x	6.85 x	6.5 x	7.5
8.9	7.7	6.5 x	6.5 x	6.5 x	7.85	6.5 x	g2
	8.7	7.5	g1 x	7.5	8.85	7.5	7.5

# The problem: low- $h$ bias

10 expanded states

29 generated states

between finding  $g_1$  and expanding right of  $S$

		8.7	7.7	6.7 x	S x	7.0	
		8.1	7.1 x				10.5
	7.9	6.9 x	6.9 x	6.9 x	6.85 x	8.0	9.0
8.9	6.7 x	6.7 x	6.7 x	6.7 x	6.85 x	6.5 x	7.5
8.9	7.7	6.5 x	6.5 x	6.5 x	7.85	6.5 x	g2
	8.7	7.5	g1 x	7.5	8.85	7.5	7.5

# Restarted search

starting from scratch

$w = 1.5$

			7.7	6.7 x	S x	7.0	
			7.1	[REDACTED]			
							g2
			g1				

# Restarted search

2 expanded state

5 generated states

before expanding right of S to find optimal path

			7.7	6.7 x	S x	7.0	
			7.1	[Blackout]			
							g2
			g1				

Continued search may be **biased** due to **early mistakes**:

- Greedy search: suboptimal area of search space
- Open list: many open states around previous goal
- Low h-value makes them look attractive
  - ⇒ Biased search explores suboptimal area in depth

Restarts overcome early mistakes of greedy search

Restarts used with **randomization** in CSPs:

- Local search (Selman et al. 1992)
- Systematic search (Gomes et al. 1998)
- Purpose: undo bad random decisions (parameter choices)  
     $\rightsquigarrow$  escape barren areas of search space

We propose restarts for a deterministic, **A\*-type** algorithm

- Purpose: undo bad greedy decisions (low- $h$  bias)
- Motivation similar to that of limited-discrepancy search (Harvey & Ginsberg 1995)

# Restarting weighted $A^*$ (RWA\*)

RWA\*: **forget** open list between iterations:

- 1 Set weight and bound
- 2 **Clear open list, (re-)start from initial state**
- 3 Conduct  $WA^*$  search, using bound for pruning
- 4 Upon new best solution: report solution, goto 1.

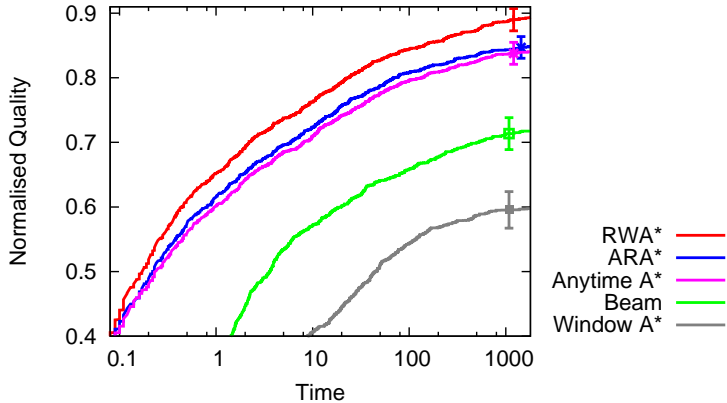
Re-use previous search effort by

- Not re-calculating h-values of states seen previously
- Remembering best known paths to states

Extra cost: re-expansions. But expansions often cheap compared to evaluations (planning: 20% vs. 80%)

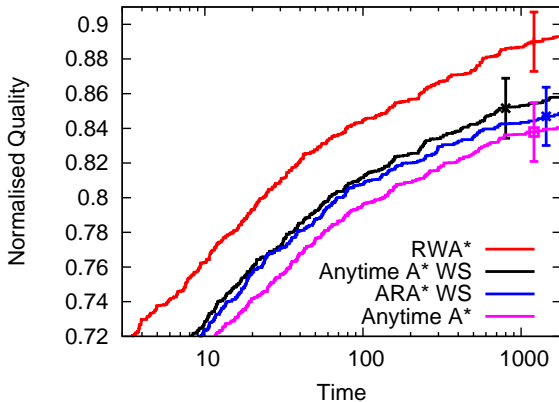
- Implemented in Fast Downward (Helmert 2006)
- Replaced greedy BFS with anytime algorithms:
  - RWA\*
  - Anytime A\*
  - ARA\*
  - Beam-stack search
  - Window A\*
- Planner-specific search enhancements used
- All 1612 classical tasks, 31 domains of previous IPCs
- Also: 3 other search benchmark domains

# Planning



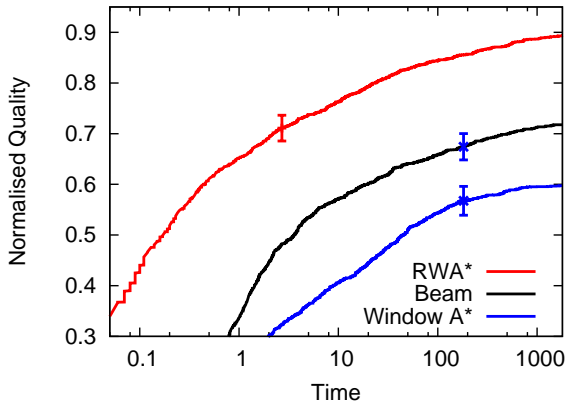
WA\* methods much better than others; RWA\* best

# Planning



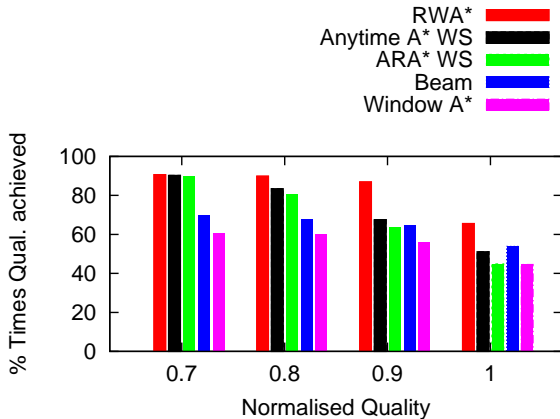
RWA\* > other WA\* methods in 40% of domains, rest on par

# Planning

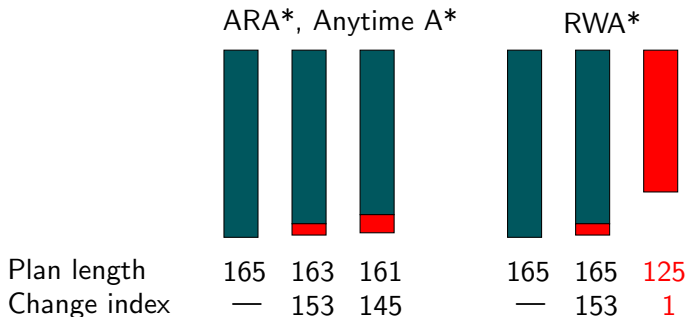


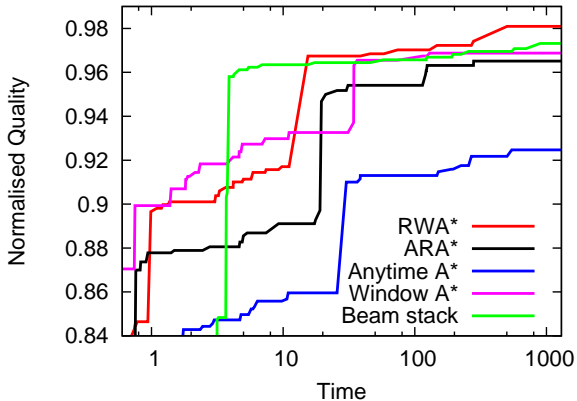
Beam-stack search, Window A\*  $>$  WA\* in some domains,  
but much worse in many other domains

# Planning



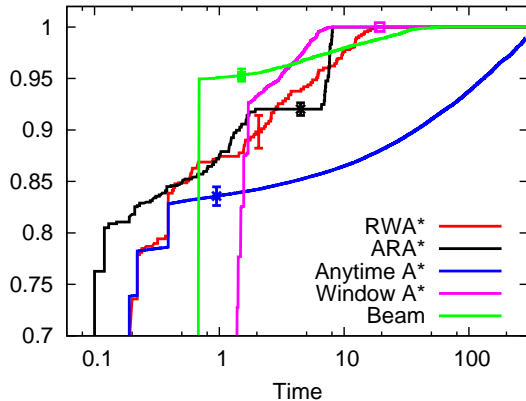
Restarts change beginning of plan rather than end (Gripper #20):





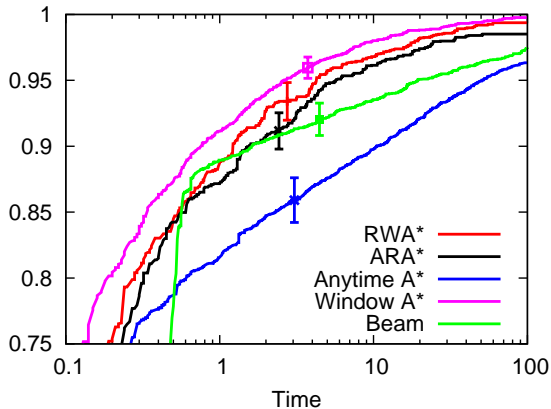
RWA\* > other WA\* methods.  
Beam-stack search and Window A\* very good here.

# Gridworld



RWA\*  $\approx$  other weight-decreasing WA\* methods.  
Beam-stack search, Window A\*: worse anytime performance.

# Sliding-tile puzzle



RWA\*  $\approx$  other weight-decreasing WA\* methods.  
Window A\* very good here.

# Summary

RWA\* dominates other methods in planning

- Restarts useful if greedy search is highly suboptimal
- E. g. if heuristics vary strongly locally

On par in other domains

- RWA\* always  $\geq$  other WA\* methods  
     $\rightsquigarrow$  even if restarts do not help, they do not hurt
- RWA\* always performs fairly well  $\rightsquigarrow$  robust,  
    while beam-stack search, Window A\* vary strongly

Undoing search effort can be worthwhile in anytime algorithms

Thank you!

Questions?