# Suboptimal and Anytime Heuristic Search on Multi-core Machines

Ethan Burns[1], Seth Lemons[1], Wheeler Ruml[1] and Rong Zhou[2]

1 University of New Hampshire

2 parc Palo Alto Research Center

[Many thanks to NSF grant IIS-0812141]

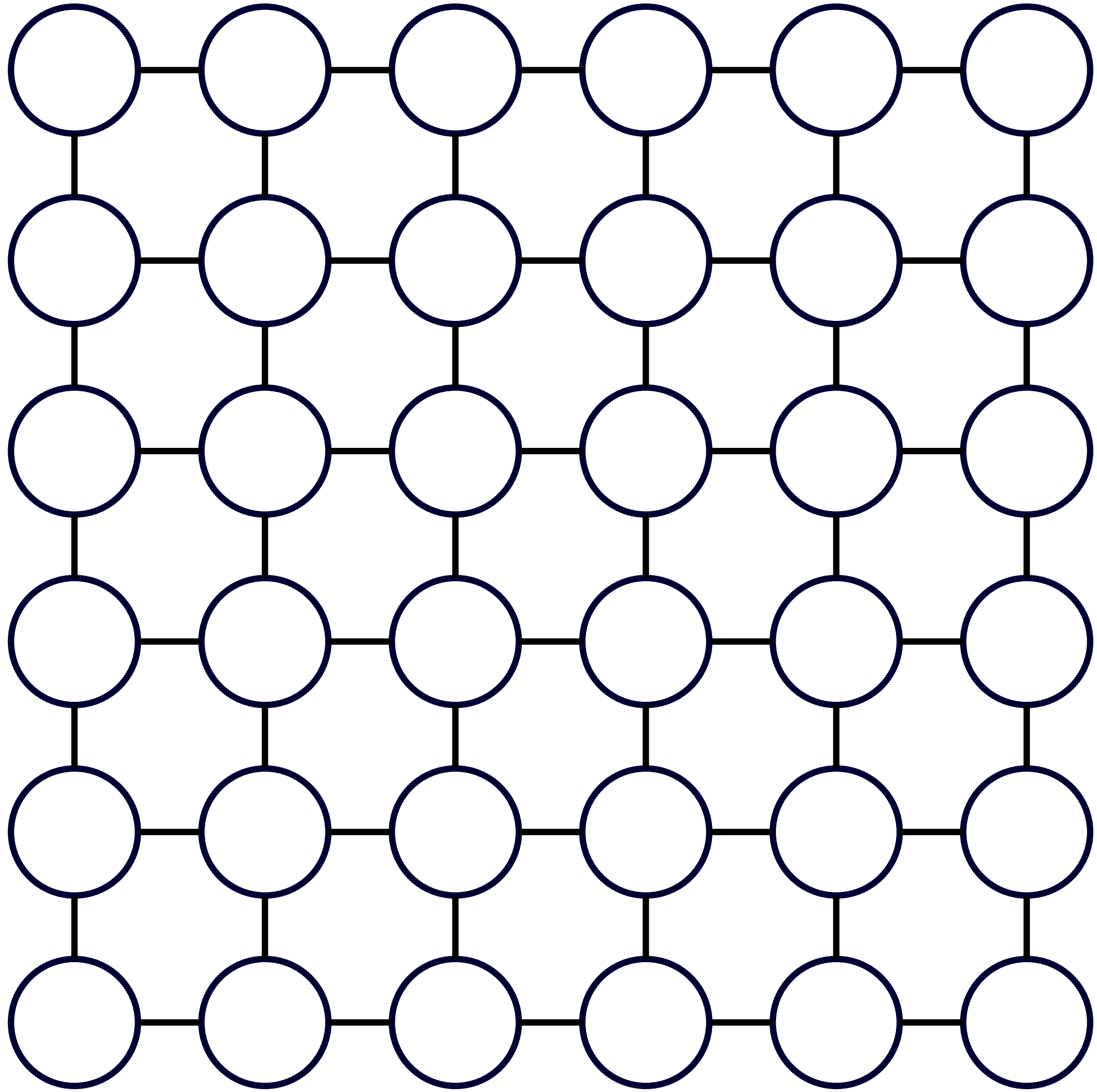
# Overview



- Background
  - Parallel Retracting A* (PRA*, Evett et al., 1995)
  - Parallel Best $N$Block First Search (PBNF, Burns et al., IJCAI 2009)
- New: Parallel bounded-suboptimal search.
  - Two pruning rules for approximate best-first suboptimal search
- New: Parallel anytime search.

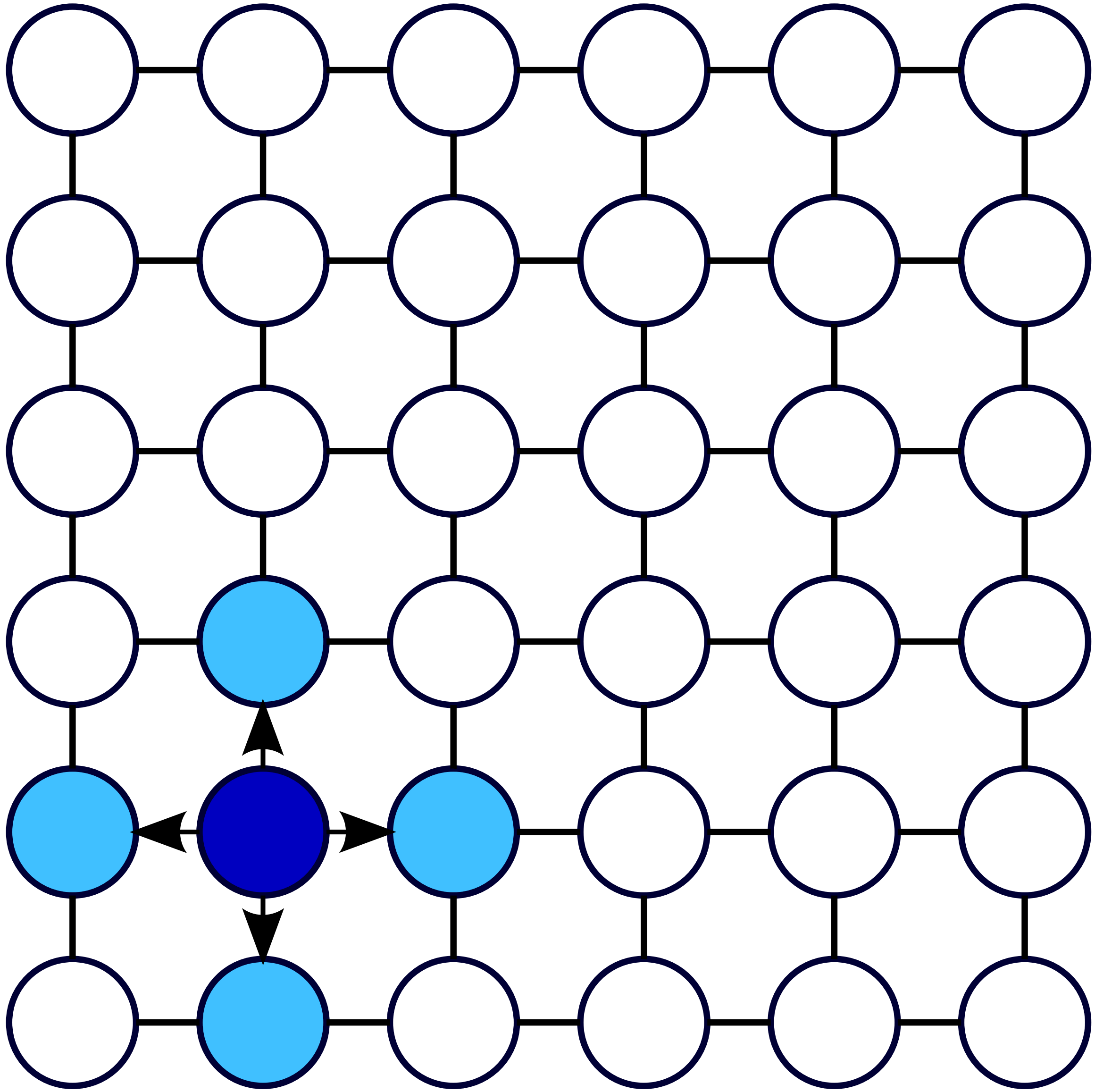
# Naive Parallel Search





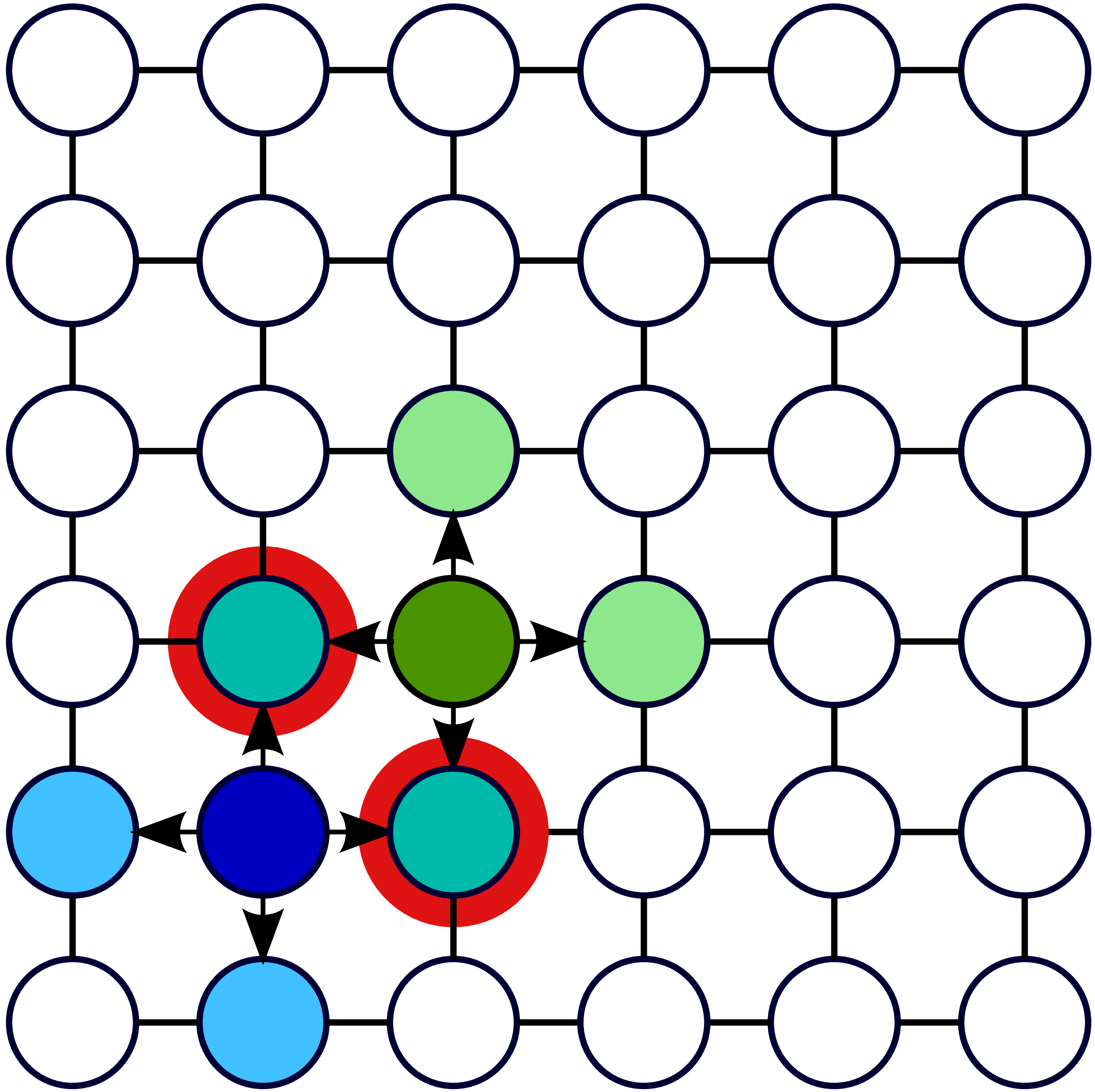
# Naive Parallel Search



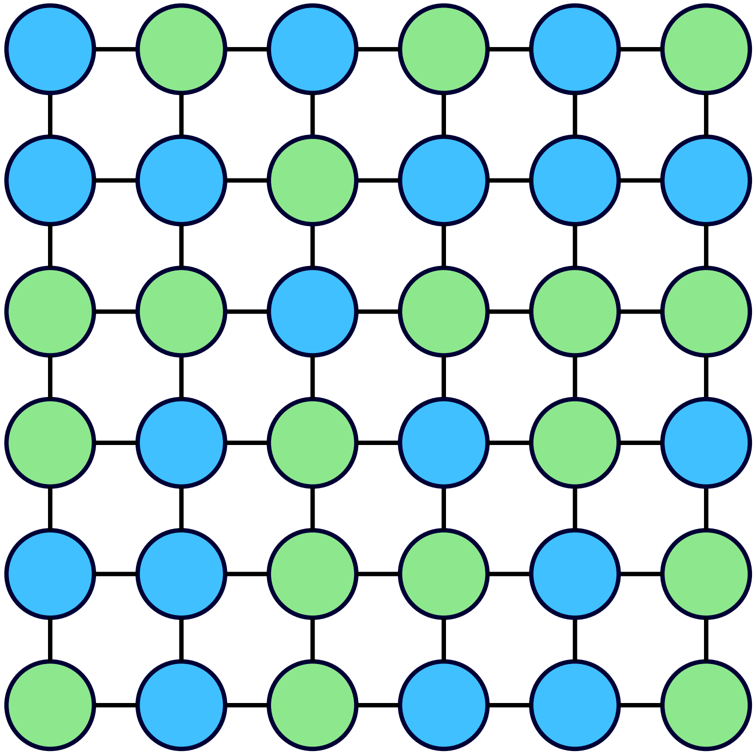


# Naive Parallel Search





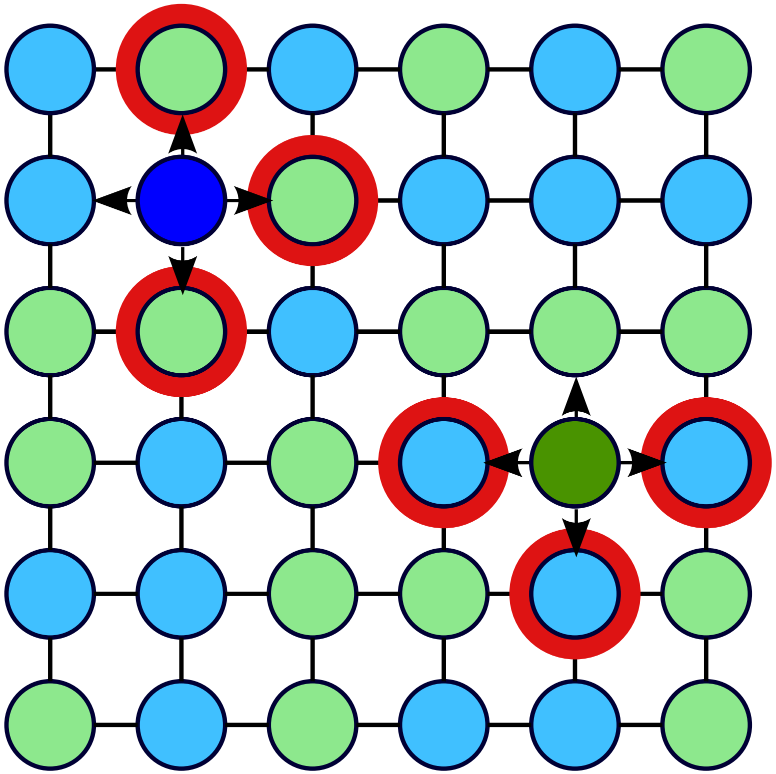
- Distribute nodes among threads using a hash function.
  - Each node has a home thread.
  - Duplicate detection can be performed locally at each thread.



# Parallel Retracting A* (PRA*, Evett et al., 1995)



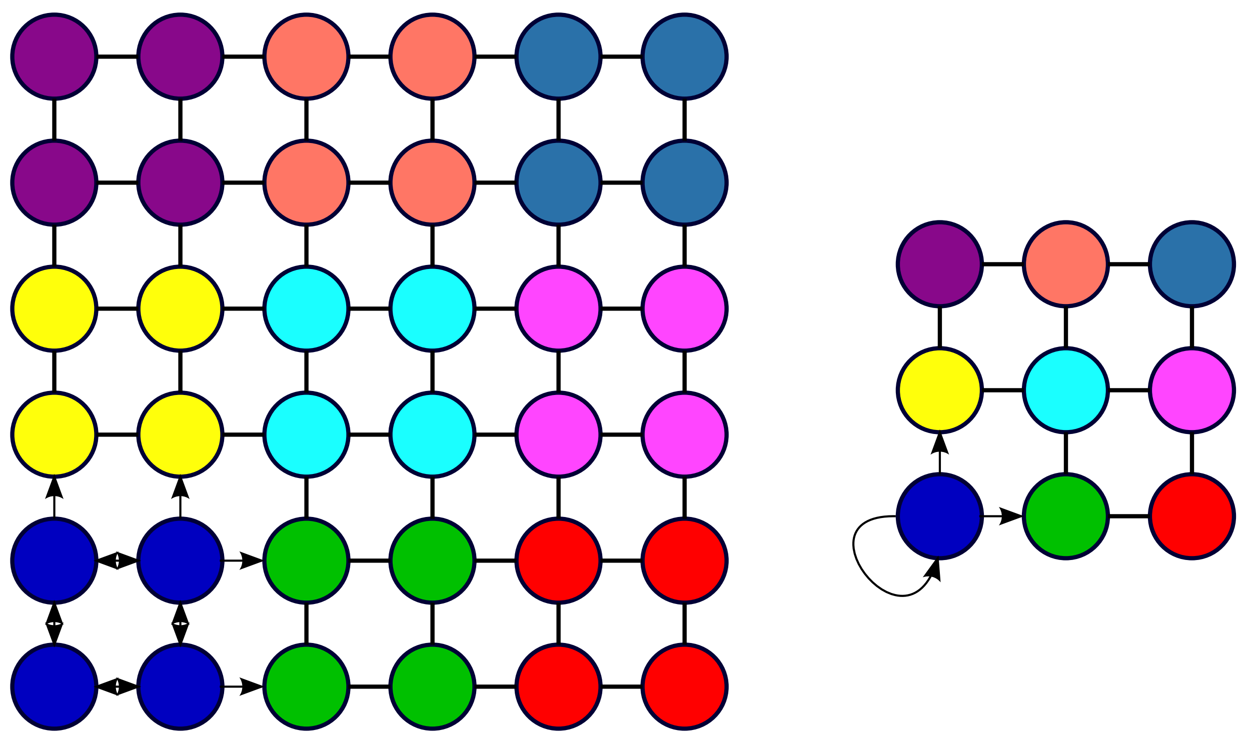
- May need to communicate nodes between threads at each generation.
- Non-blocking: HDA* (Kishimoto et al., best paper award ICAPS 2009)



# Parallel Best $N$block First (PBNF, Burns et al., 2009)



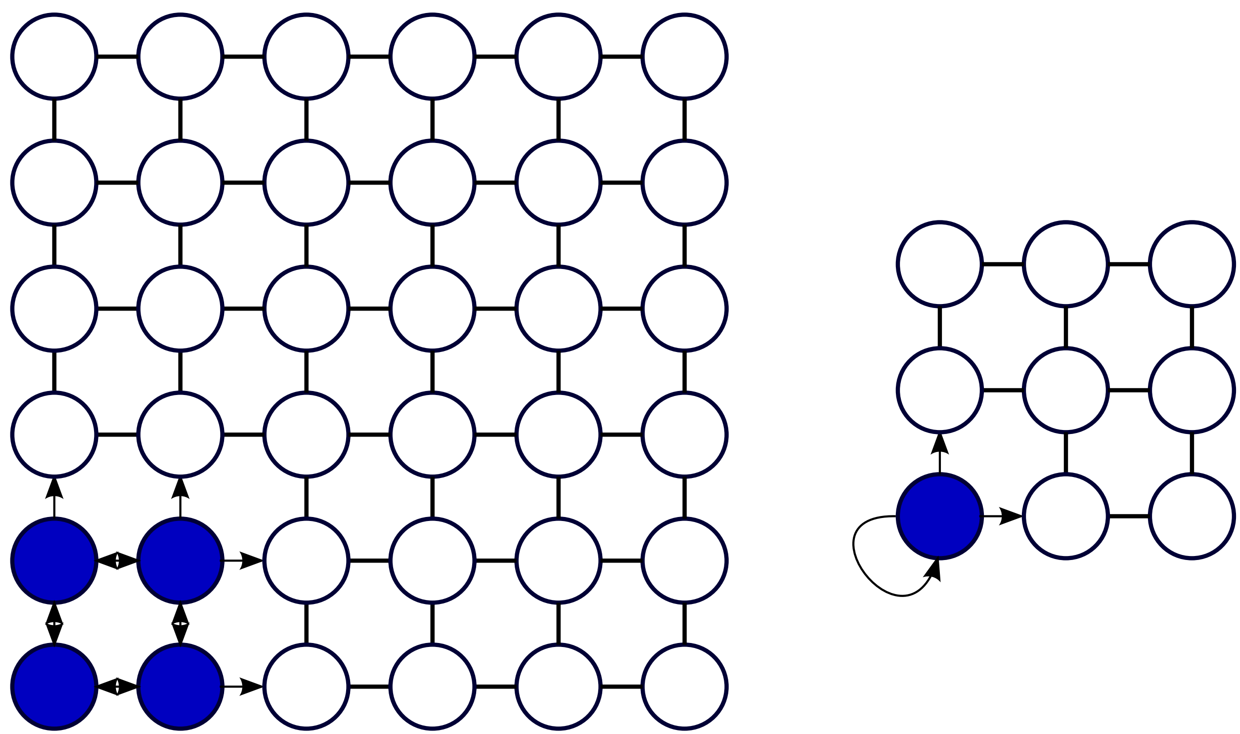
- Work is divided among threads using a special hash function based on abstraction. (Zhou and Hansen, 2007)
  - Few possible destinations for children.



# Parallel Best $N$block First (PBNF, Burns et al., IJCAI 2009)



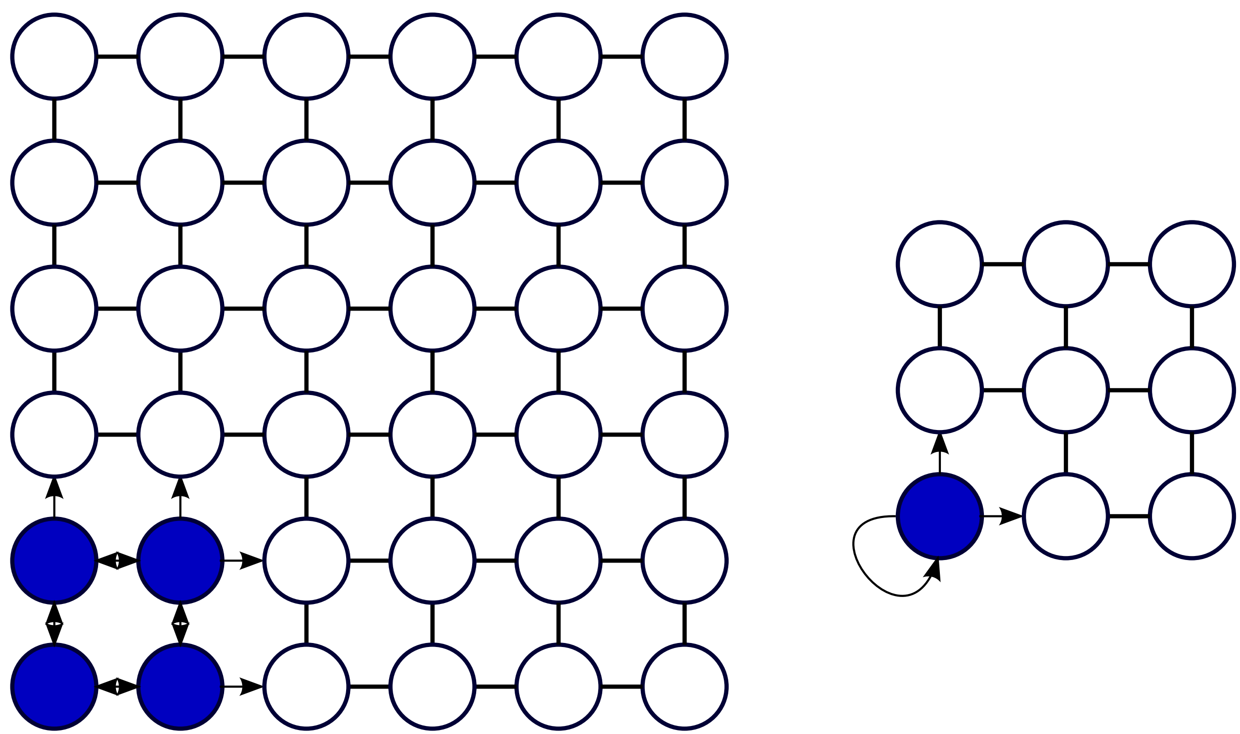
- Work is divided among threads using a special hash function based on abstraction.
  - Threads search groups of nodes called $n$blocks.



# Parallel Best $N$block First (PBNF, Burns et al., IJCAI 2009)



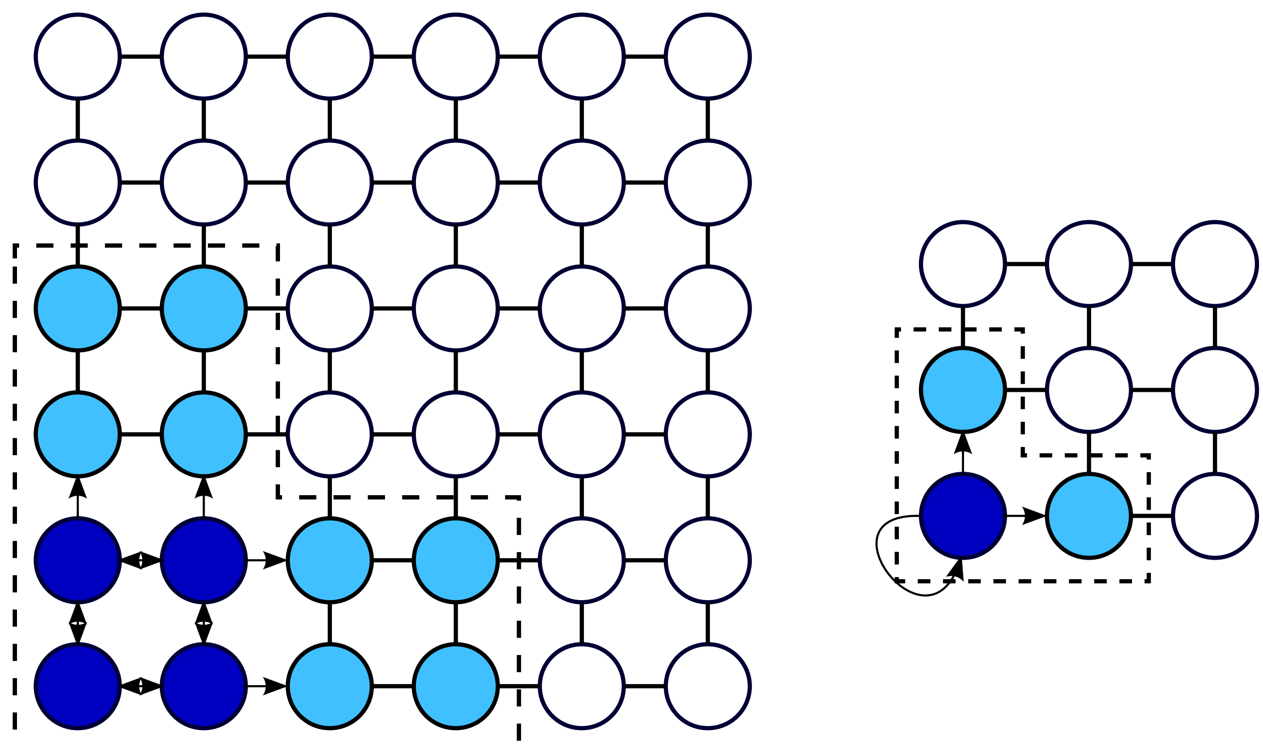
- Work is divided among threads using a special hash function based on abstraction.
  - $n$blocks have an open and closed list.



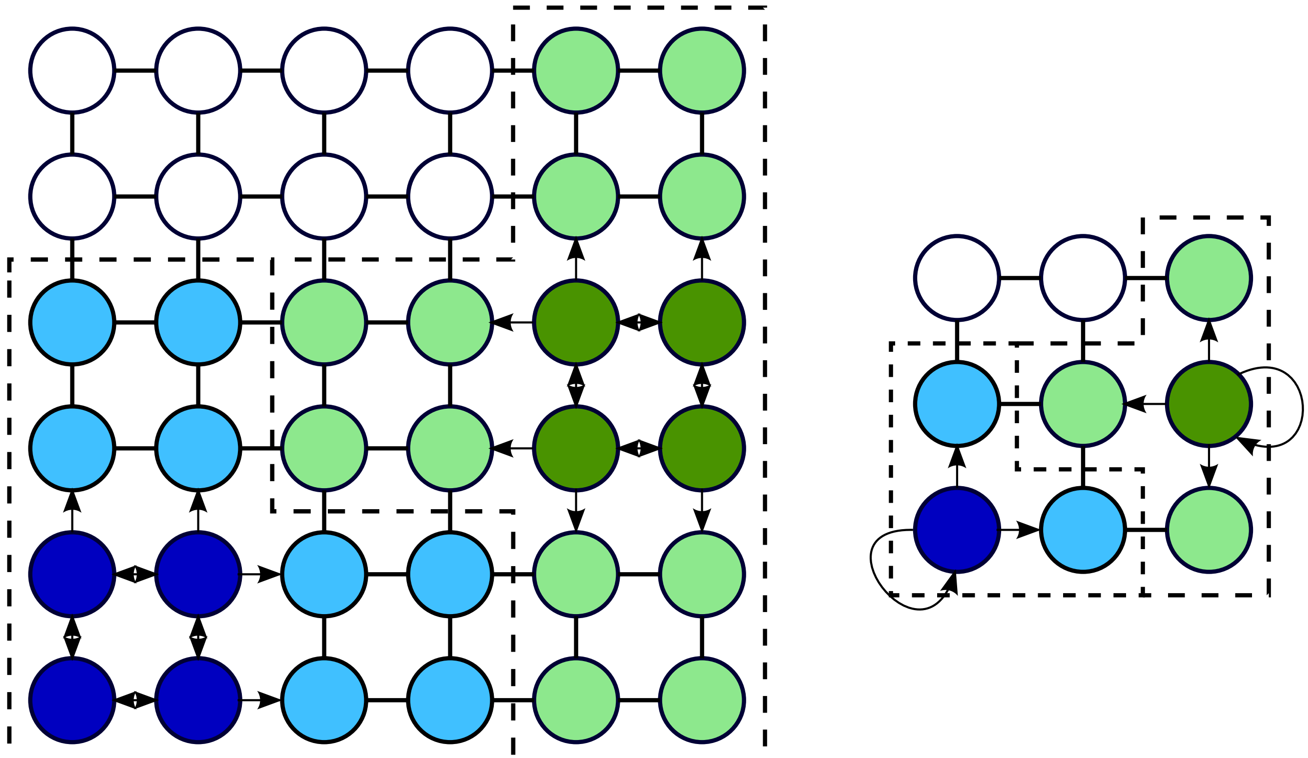
# Parallel Best $N$block First (PBNF, Burns et al., IJCAI 2009)



- Work is divided among threads using a special hash function based on abstraction.
  - An $n$block and its successors: *duplicate detection scope*.



- Work is divided among threads using a special hash function based on abstraction.
  - *Disjoint* duplicate detection scopes searched in parallel.



# Parallel Best $N$Block First (PBNF, Burns et al., IJCAI 2009)



1. Search disjoint $n$blocks in parallel.
   - Maintain a heap of free $n$blocks.
   - Greedily acquire best free $n$block (and its scope).
2. Each $n$block is searched in $f(n) = g(n) + h(n)$ order.
   - Switch $n$blocks when a better one becomes free.
   - Approximates best-first order.
3. Stop when the incumbent solution is optimal.
   - Prune nodes on the cost of the incumbent
   - Incumbent is optimal when all nodes are pruned.
4. See paper for proof of correctness (no livelock).

# Optimal Search (New since paper)

# Algorithms Not Shown



**Parallel A***

- Basic A* with a lock on open and closed lists.

**Lock-free PA***

- PA* with lock-free data structures.

**KBFS** (Felner et al., 2003)

- Expand the $K$ best open nodes in parallel.

**PSDD** (Zhou and Hansen, 2007)

- Abstraction to find disjoint portions of a search space.
- Breadth-first search
- All threads synchronize at each layer

**IDPSDD**

- PSDD with iterative-deepening for bounds.

**BFPSDD***

- PSDD, but search in $f(n)$ layers.

# Algorithms



**PRA*** (Evett et al., 1995)

- Distributes nodes with a hash function.

**HDA*** (Kishimoto et al., ICAPS 2009)

- PRA* with non-blocking communication.
- Originally developed for distributed memory.

**APRA* and AHDA***

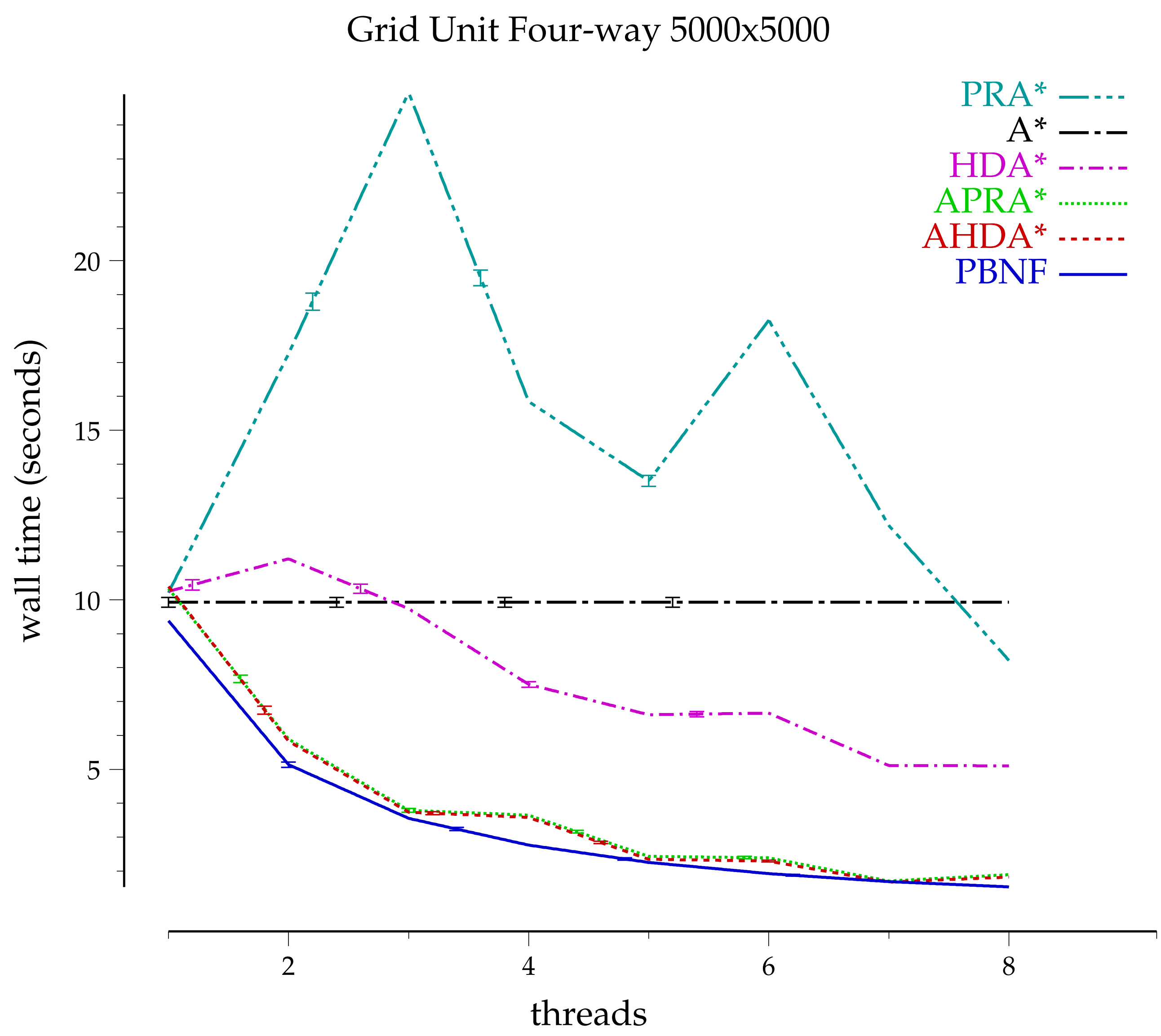
- PRA* and HDA* with abstraction based hashing function.

**PBNF** (Burns et al., IJCAI 2009)

- Uses abstraction to decompose the search space.
- Greedily acquire best free nodes.

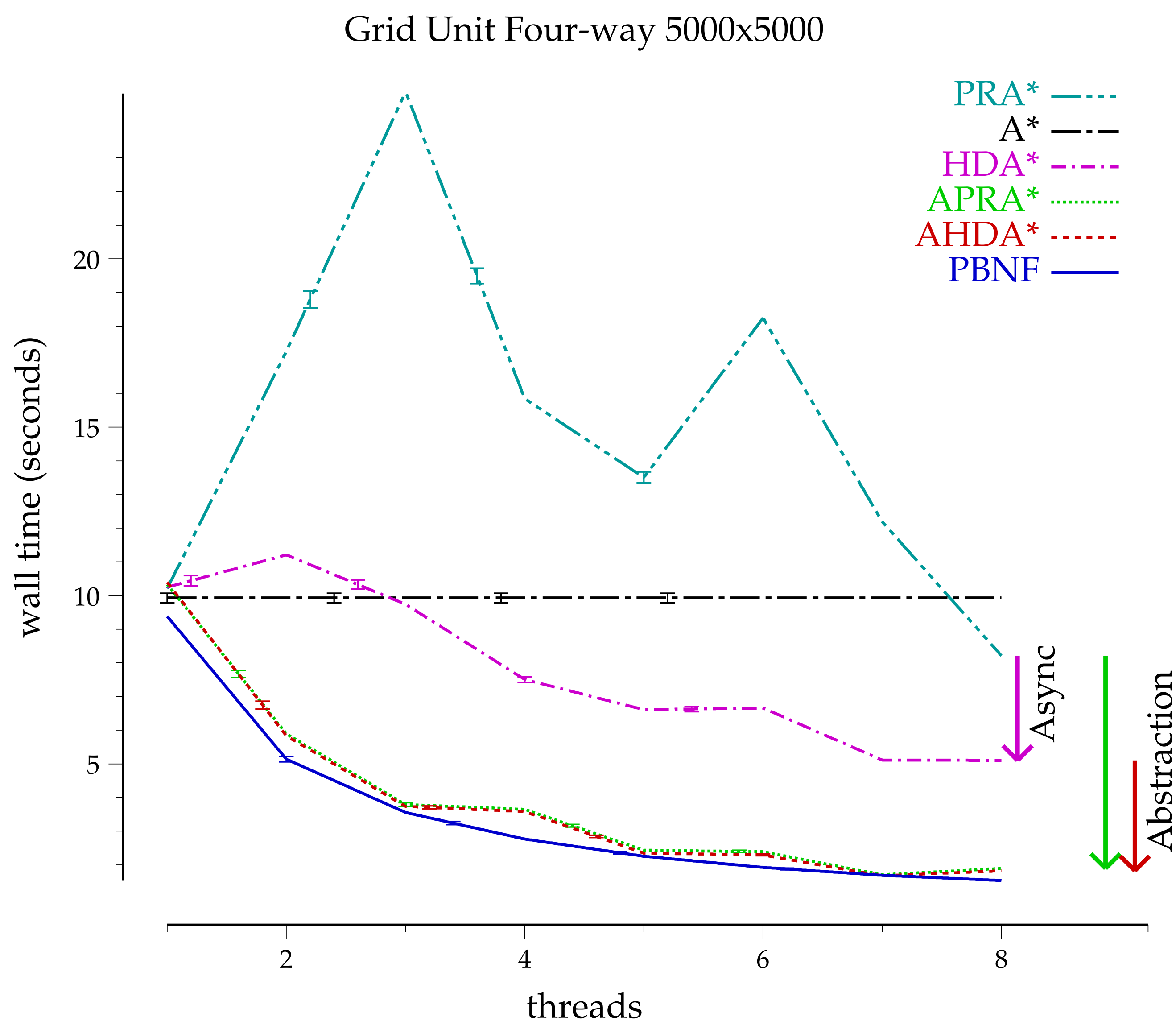
# Four-way Grid Pathfinding 5000x5000





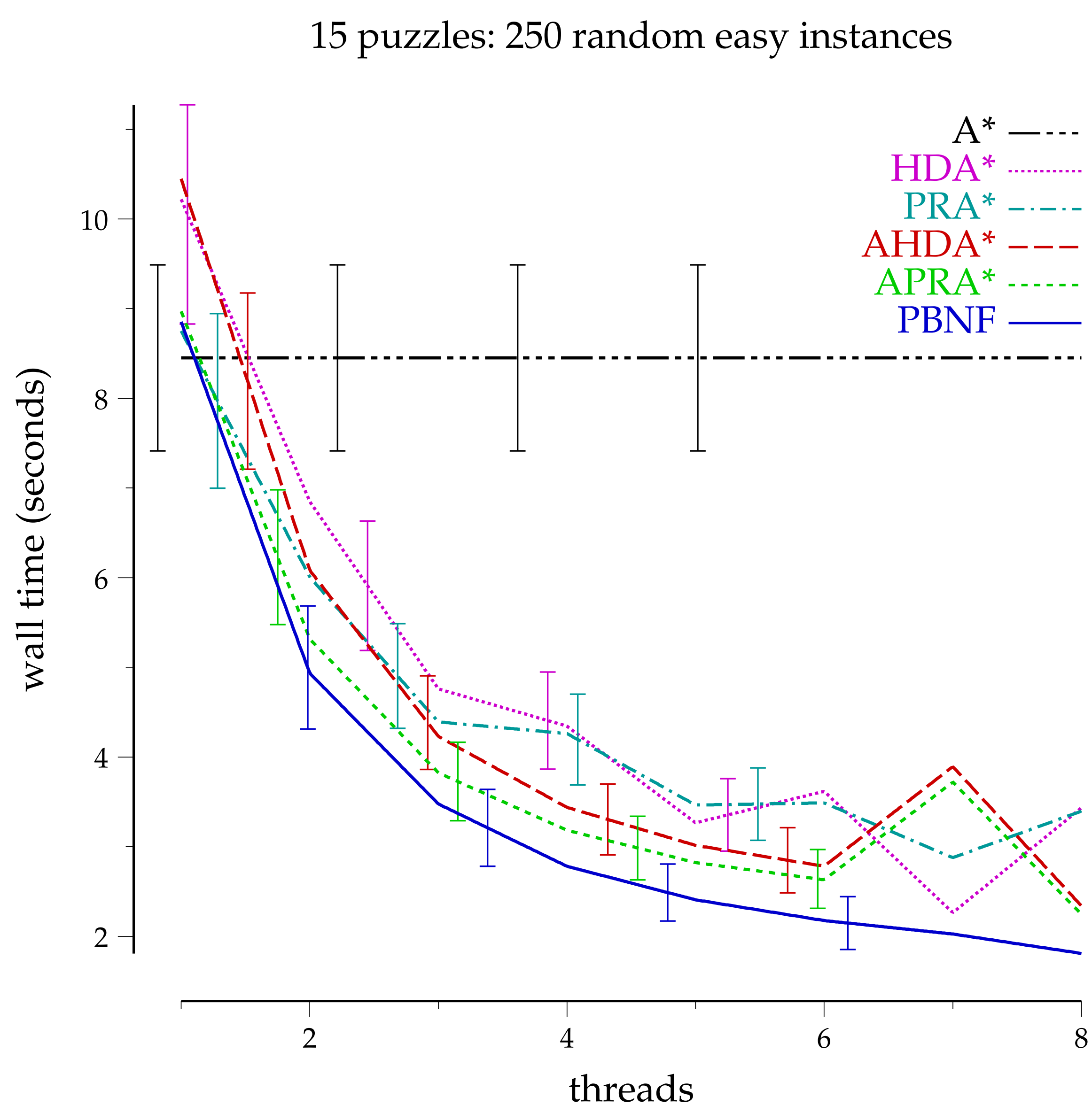
# Four-way Grid Pathfinding 5000x5000





# Easy 15-Puzzles





# Optimal STRIPS Planning



| | threads | logistics-6 | blocks-14 | gripper-7 | satellite-6 | elevator-12 | freecell-3 | depots-7 | driverlog-11 | gripper-8 |
|---|---|---|---|---|---|---|---|---|---|---|
| A* | 1 | 2.30 | 5.19 | 117.78 | 130.85 | 335.74 | 199.06 | M | M | M |
| APRA* | 1 | 1.44 | 7.37 | 62.61 | 95.11 | 215.19 | 153.71 | 319.48 | 334.28 | 569.26 |
| | 3 | 0.75 | 5.30 | 43.13 | 42.85 | 243.24 | 122.00 | 138.30 | 99.37 | 351.87 |
| | 5 | 1.09 | 3.26 | 37.62 | 67.38 | 211.45 | 63.47 | 67.24 | 89.73 | 236.93 |
| | 7 | 0.81 | 2.92 | 26.78 | 52.82 | 169.92 | 37.94 | 49.58 | 104.87 | 166.19 |
| AHDA* | 1 | 1.44 | 7.13 | 59.51 | 95.50 | 206.16 | 147.96 | 299.66 | 315.51 | 532.51 |
| | 3 | 0.70 | 5.07 | 33.95 | 33.59 | 96.82 | 93.55 | 126.34 | 85.17 | 239.22 |
| | 5 | 0.48 | 2.25 | 15.97 | 24.11 | 67.68 | 38.24 | 50.97 | 51.28 | 97.61 |
| | 7 | **0.40** | 2.13 | 12.69 | 18.24 | 57.10 | **27.37** | 39.10 | 48.91 | 76.34 |
| PBNF | 1 | 1.17 | 6.21 | 39.58 | 77.02 | 150.39 | 127.07 | 156.36 | 154.15 | 235.46 |
| | 3 | 0.64 | 2.69 | 16.87 | 24.09 | 53.45 | 47.10 | 63.04 | 59.98 | 98.21 |
| | 5 | 0.56 | 2.20 | 11.23 | 17.29 | 34.23 | 38.07 | 42.91 | 38.84 | 63.65 |
| | 7 | 0.62 | **2.02** | **9.21** | **13.67** | **27.02** | 37.02 | **34.66** | **31.22** | **51.50** |

Wall time in seconds

# Summary of Optimal Results



- PBNF gave the best performance and scalability across all domains tested.
- Non-blocking communication improved the performance of PRA*, confirming results from (Kishimoto et al., 2009).
- Abstraction improved the performance of PRA* and HDA*.

# Bounded Suboptimal Search

# Bounded suboptimal



- Simple to convert PRA* and PBNF to bounded suboptimal
  - Sort open lists on $f'(n) = g(n) + w \cdot h(n)$.
  - Stop when $\min_{n \in open} w \cdot f(n) \geq g(s)$.
  - Two new pruning rules: see paper.
- Suboptimal PBNF
  - Sort $n$block free-list on $\min_{n \in open} f'(n)$.

# Four-way Grid Pathfinding 5000x5000



| | weight | threads 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| wPBNF | 1.1 | 0.84 | **1.51** | **2.23** | **2.87** | **3.41** | **4.02** | **4.55** | **5.03** |
| | 1.2 | 0.77 | 1.42 | 2.09 | 2.69 | 3.24 | 3.72 | 4.12 | 4.52 |
| | 1.4 | 0.42 | 0.92 | 1.39 | 1.83 | 2.31 | 2.51 | 2.77 | 2.98 |
| | 1.8 | 0.62 | 0.72 | 0.81 | 0.82 | 0.83 | 0.86 | 0.85 | 0.87 |
| | 3.4 | 0.71 | 0.69 | 0.69 | 0.69 | 0.67 | 0.65 | 0.64 | 0.64 |
| wAHDA* | 1.1 | **0.87** | 1.41 | 2.04 | 1.82 | 2.74 | 3.40 | 4.09 | 3.57 |
| | 1.2 | 0.79 | 1.22 | 1.82 | 1.75 | 3.28 | 3.29 | 3.96 | 3.48 |
| | 1.4 | 0.31 | 0.69 | 1.51 | 1.55 | 2.62 | 2.47 | 3.05 | 2.68 |
| | 1.8 | 0.55 | 0.74 | 0.94 | 0.69 | 0.83 | 0.81 | 0.74 | 0.64 |
| | 3.4 | 0.71 | 0.69 | 0.73 | 0.51 | 0.59 | 0.59 | 0.56 | 0.48 |

Speedup over serial wA

- wPBNF gave the best performance at all but 1 thread.
- Lower weight gives more speedup.

# Korf's 100 15-Puzzles



| | weight | threads 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| wPBNF | 1.4 | 0.86 | 1.40 | **2.27** | 2.01 | **2.41** | 2.48 | **2.68** | **2.58** |
| | 1.7 | 0.98 | 1.34 | 1.70 | 1.87 | 2.33 | **2.63** | 2.33 | 2.08 |
| | 2.0 | 0.96 | 1.17 | 1.45 | 1.44 | 1.57 | 1.48 | 1.56 | 1.48 |
| | 3.0 | **1.09** | 1.34 | 1.46 | 1.44 | 1.41 | 1.34 | 1.38 | 1.21 |
| | 5.0 | 0.93 | 1.04 | 1.12 | 1.04 | 1.07 | 1.13 | 0.99 | 0.92 |
| wAHDA* | 1.4 | 0.84 | 1.50 | 1.90 | **2.33** | 2.37 | 2.39 | 2.39 | 2.47 |
| | 1.7 | 0.82 | 1.42 | 1.66 | 1.90 | 1.68 | 1.75 | 1.64 | 1.70 |
| | 2.0 | 0.80 | **1.52** | 1.48 | 1.74 | 1.44 | 1.23 | 1.25 | 1.23 |
| | 3.0 | 0.75 | 1.39 | 1.30 | 1.31 | 1.10 | 0.88 | 0.73 | 0.70 |
| | 5.0 | 0.71 | 1.11 | 0.91 | 0.85 | 0.70 | 0.54 | 0.45 | 0.43 |

Speedup over serial wA

- wPBNF often gave the best performance.
- Lower weight gives more speedup.

# STRIPS Planning



| | | wAPRA* | | | | wAHDA* | | | | wPBNF | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1.5 | 2 | 3 | 5 | 1.5 | 2 | 3 | 5 | 1.5 | 2 | 3 | 5 |
| 2 threads | logistics-8 | 0.99 | 1.02 | 0.59 | 1.37 | 1.25 | 1.11 | 0.80 | 1.51 | 2.68 | 2.27 | **4.06** | 1.00 |
| | blocks-16 | 1.29 | 0.88 | 4.12 | 0.30 | 1.52 | 1.09 | **4.86** | 0.38 | 0.93 | 0.54 | 0.48 | 1.32 |
| | gripper-7 | 0.76 | 0.76 | 0.77 | 0.77 | 1.36 | 1.35 | 1.33 | 1.30 | 2.01 | 1.99 | 1.99 | **2.02** |
| | satellite-6 | 0.68 | 0.93 | 0.70 | 0.75 | 1.15 | 1.09 | 1.28 | 1.44 | 2.02 | 1.53 | **5.90** | 3.04 |
| | elevator-12 | 0.65 | 0.72 | 0.71 | 0.77 | 1.16 | 1.20 | 1.27 | 1.22 | 2.02 | 2.08 | **2.21** | 2.15 |
| | freecell-3 | 1.03 | 1.00 | 1.78 | 1.61 | 1.49 | 1.20 | 7.56 | 1.40 | 2.06 | 0.84 | 8.11 | **10.69** |
| | depots-13 | 0.73 | 1.25 | 0.97 | 1.08 | 0.92 | 1.29 | 0.96 | 1.09 | 2.70 | **4.49** | 0.82 | 0.81 |
| | driverlog-11 | 0.91 | 0.79 | 0.94 | 0.93 | **1.30** | 0.97 | 0.96 | 0.93 | 0.85 | 0.19 | 0.69 | 0.62 |
| | gripper-8 | 0.63 | 0.61 | 0.62 | 0.62 | 1.14 | 1.16 | 1.15 | 1.16 | 2.06 | 2.04 | **2.08** | 2.07 |
| 7 threads | logistics-8 | 3.19 | 3.10 | 3.26 | 2.58 | 4.59 | 4.60 | 3.61 | 2.58 | **7.10** | 6.88 | 1.91 | 0.46 |
| | blocks-16 | 3.04 | 1.37 | 1.08 | 0.37 | **3.60** | 1.62 | 0.56 | 0.32 | 2.87 | 0.70 | 0.37 | 1.26 |
| | gripper-7 | 1.71 | 1.74 | 1.73 | 1.82 | 3.71 | 3.66 | 3.74 | 3.83 | **5.67** | 5.09 | 5.07 | 5.18 |
| | satellite-6 | 1.11 | 1.01 | 1.29 | 1.44 | 3.22 | 3.57 | 3.05 | 3.60 | 4.42 | 2.85 | 2.68 | **5.89** |
| | elevator-12 | 0.94 | 0.97 | 1.04 | 1.02 | 2.77 | 2.88 | 2.98 | 3.03 | 6.32 | 6.31 | 6.60 | **7.10** |
| | freecell-3 | 3.09 | 7.99 | 2.67 | 2.93 | 4.77 | 2.71 | 48.66 | 4.77 | 7.01 | 2.31 | 131.12 | **1,721.33** |
| | depots-13 | 2.38 | 5.36 | 1.13 | 1.17 | 2.98 | **6.09** | 1.22 | 1.17 | 3.12 | 1.80 | 0.87 | 0.88 |
| | driverlog-11 | 1.90 | 1.25 | 0.93 | 0.92 | **3.52** | 1.48 | 0.95 | 0.92 | 1.72 | 0.43 | 0.67 | 0.42 |
| | gripper-8 | 1.70 | 1.68 | 1.68 | 1.74 | 3.71 | 3.63 | 3.67 | 4.00 | **5.85** | 5.31 | 5.40 | 5.44 |

Speedup over serial wA*

- Most **red** is under wPBNF (13 of 18).
- Blue is everywhere.

- In general speedup was not as good as optimal search.
  - Some harder problems gave excellent speedup.
- Lower weights can increase benefit of parallelizing.

# Anytime Search

# Anytime



- Simple to convert PRA* and PBNF to anytime.
  - Sort open lists on $f'(n) = g(n) + w \cdot h(n)$.
  - Stop when $\min_{n \in open} f(n) \geq g(s)$ (same as optimal).
- Anytime PBNF
  - Sort $n$block free-list on $\min_{n \in open} f'(n)$.
- Parallel analogue to Anytime Weighted A* (Hansen and Zhou, JAIR 2007)

| | | AwAPRA* 1.5 | AwAPRA* 2 | AwAPRA* 3 | AwAPRA* 5 | AwAHDA* 1.5 | AwAHDA* 2 | AwAHDA* 3 | AwAHDA* 5 | AwPBNF 1.5 | AwPBNF 2 | AwPBNF 3 | AwPBNF 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 threads | logistics-6 | 1.09 | 1.06 | 1.40 | 1.40 | 1.23 | 1.21 | 1.59 | 1.66 | 1.06 | 1.35 | 1.94 | **1.98** |
| | blocks-14 | 1.36 | 7.76 | 56.41 | 90.16 | 1.62 | 9.90 | 63.60 | **110.16** | 1.91 | 1.99 | 13.22 | 22.36 |
| | gripper-7 | 0.78 | 0.77 | 0.76 | 0.75 | 1.35 | 1.33 | 1.32 | 1.33 | **2.05** | 1.96 | 1.99 | 1.95 |
| | satellite-6 | 0.77 | 0.78 | 0.78 | 0.76 | 1.26 | 1.23 | 1.24 | 1.23 | 1.58 | 1.96 | **1.98** | 1.91 |
| | elevator-12 | 0.64 | 0.67 | 0.69 | 0.70 | 1.20 | 1.19 | 1.16 | 1.17 | 2.01 | 2.07 | **2.13** | 2.07 |
| | freecell-3 | 1.37 | 1.43 | 4.61 | 1.37 | 1.66 | 1.68 | 5.65 | 1.95 | 1.93 | 1.06 | 2.78 | **6.23** |
| | depots-7 | 1.24 | 1.30 | 1.30 | 2.68 | 1.51 | 1.51 | 1.50 | 3.18 | 1.94 | 2.00 | 2.01 | **4.10** |
| | driverlog-11 | 1.15 | 1.19 | 1.11 | 1.20 | 1.50 | 1.55 | 1.46 | 1.54 | 1.95 | **2.10** | 1.99 | 0.77 |
| | gripper-8 | 0.61 | 0.62 | 0.62 | 0.62 | 1.16 | 1.11 | 1.14 | 1.11 | 2.04 | 2.05 | **2.09** | 2.06 |
| 7 threads | logistics-6 | 1.45 | 1.43 | 1.81 | 1.81 | 2.87 | 2.81 | 3.65 | 3.74 | 2.04 | 2.46 | 4.19 | **4.21** |
| | blocks-14 | 2.54 | 15.63 | 98.52 | 177.08 | 3.30 | 19.91 | 132.97 | **231.45** | 3.72 | 22.37 | 25.69 | 7.20 |
| | gripper-7 | 1.77 | 1.68 | 1.71 | 1.73 | 3.75 | 3.69 | 3.61 | 3.67 | **5.61** | 5.05 | 5.03 | 5.06 |
| | satellite-6 | 1.22 | 1.22 | 1.26 | 1.26 | 3.56 | 3.46 | 3.51 | 3.50 | **5.96** | 4.66 | 5.74 | 4.70 |
| | elevator-12 | 0.93 | 0.93 | 0.95 | 0.94 | 2.77 | 2.75 | 2.79 | 2.77 | 6.18 | 6.03 | **6.20** | 6.05 |
| | freecell-3 | 3.64 | 3.75 | 11.59 | 4.44 | 5.00 | 4.97 | 16.36 | **21.57** | 3.54 | 1.50 | 15.32 | 11.46 |
| | depots-7 | 3.60 | 3.64 | 3.65 | 7.60 | 4.41 | 4.42 | 4.40 | 9.25 | 5.74 | 5.52 | 5.48 | **10.84** |
| | driverlog-11 | 3.04 | 3.20 | 3.05 | 3.17 | 4.74 | 4.82 | 4.66 | 4.87 | 5.78 | **5.83** | 5.73 | 2.18 |
| | gripper-8 | 1.72 | 1.67 | 1.70 | 1.69 | 3.65 | 3.61 | 3.66 | 3.66 | **5.82** | 5.36 | 5.39 | 5.39 |

Speedup over serial AwA* (to convergence)

# Summary of Anytime Results



- Outperforms serial anytime search.
- AwPBNF gave the best performance on all but three domains.
- AwAHDA* occasionally gave much better performance.

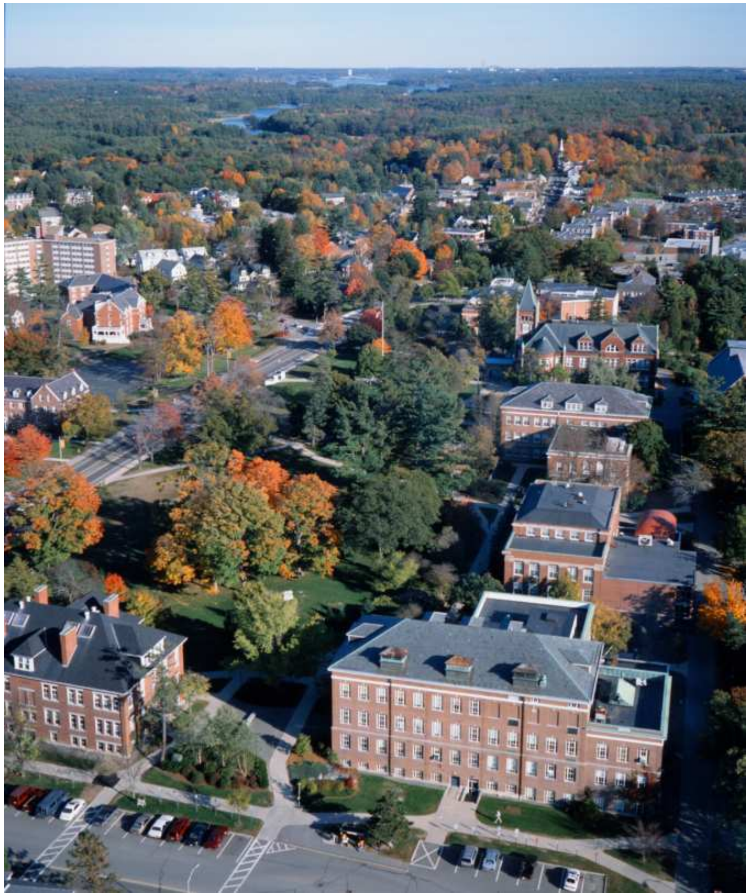
# Conclusion

# Conclusion



- Parallel search can make your programs run faster today.
  - Multi-core is not going away.
  - Email me for the code (C++): burns.ethan@gmail.com
- PBNF and PRA* are simple and general.
  - Easily extendable to weighted and anytime search.
  - PBNF generally performed better than the other algorithms tested.
- Abstraction is beneficial for parallel search.
- Parallel search is more beneficial on harder problems.

Tell your students to apply to grad school in CS at UNH!

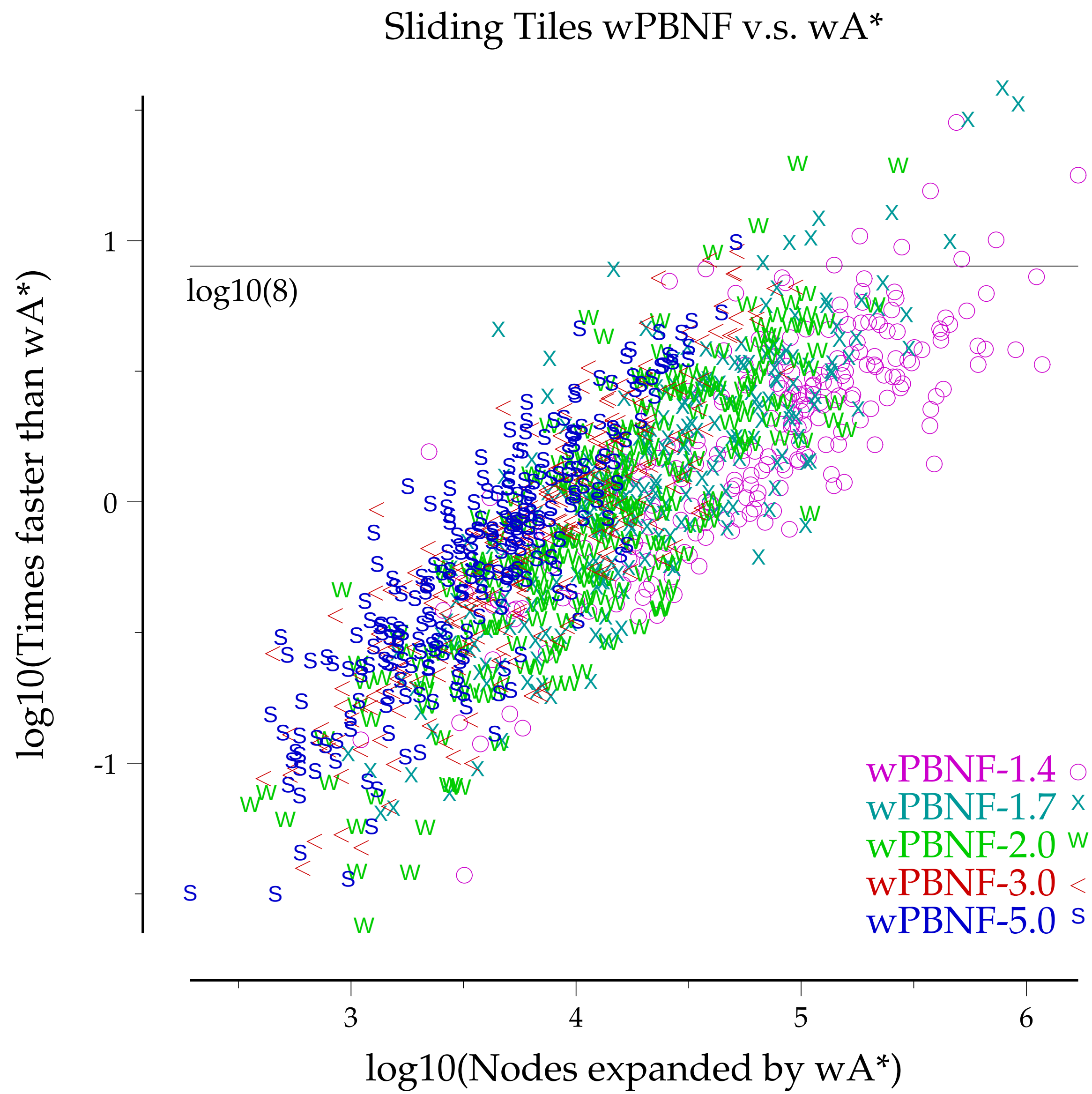


- friendly faculty
- funding
- individual attention
- beautiful campus
- low cost of living
- easy access to Boston, White Mountains
- strong in AI, infoviz, networking, systems

# Additional Slides

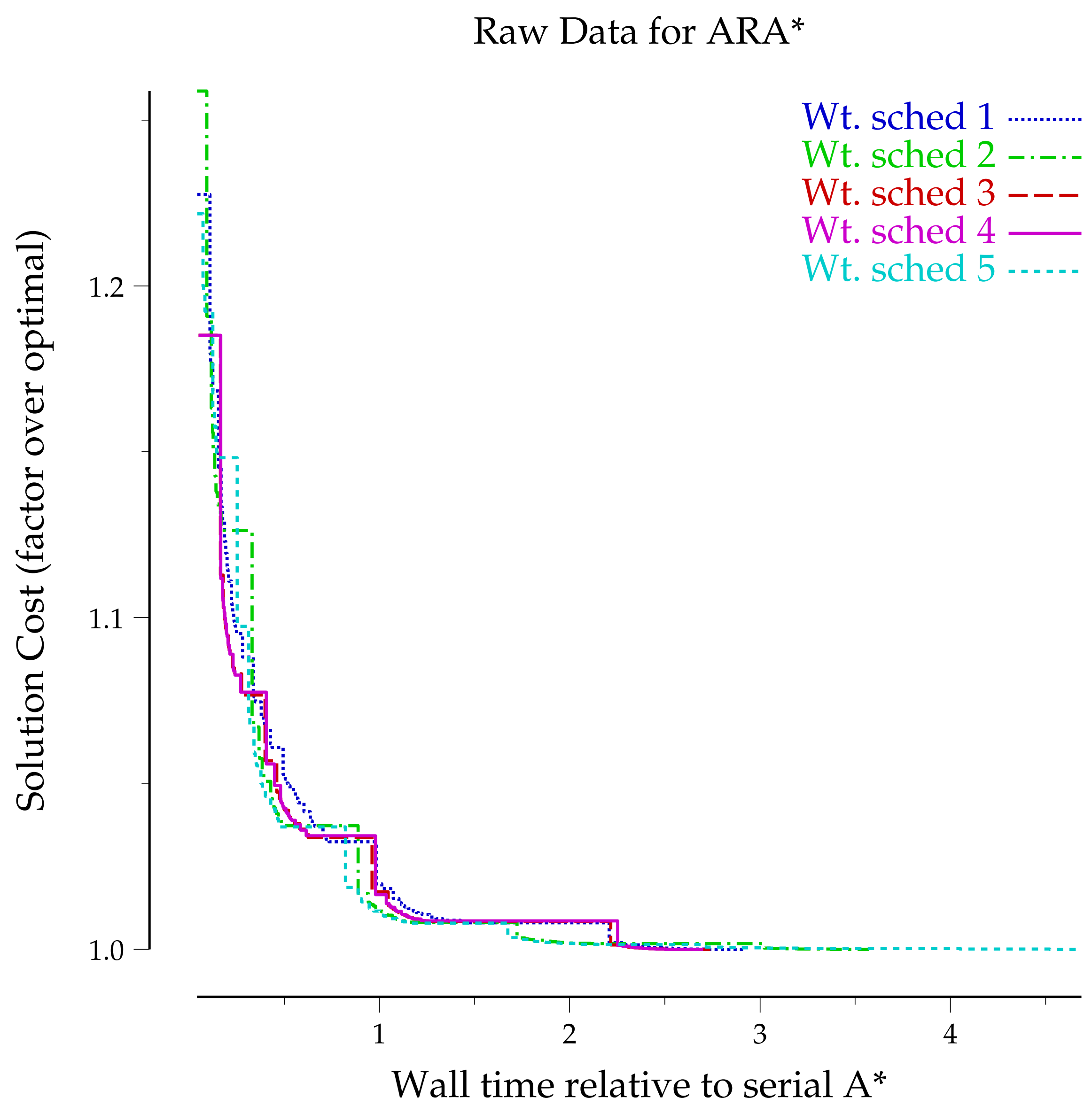
# Difficulty versus Advantage over wA*





# Hull Plots

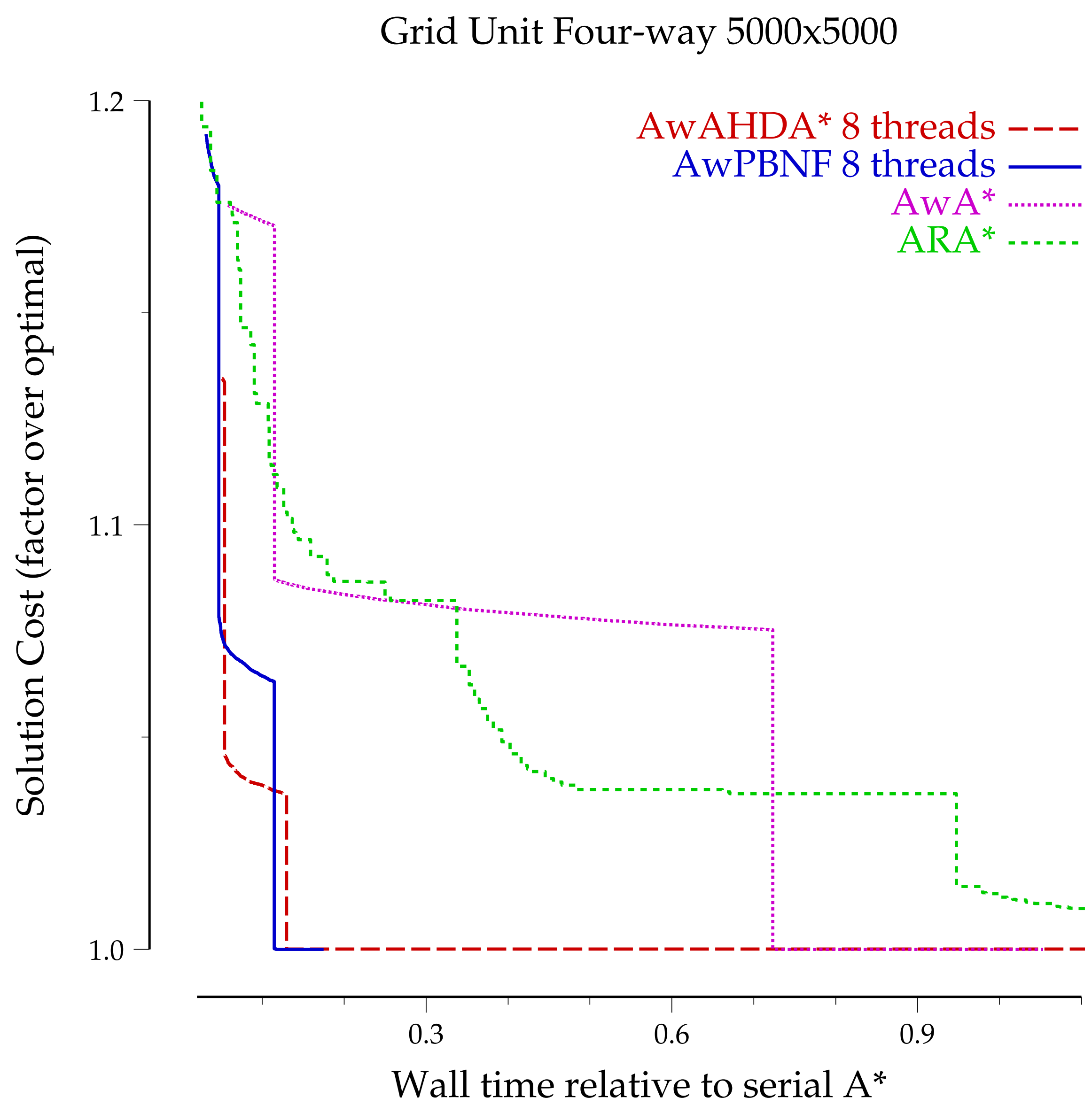




Raw Data for ARA*
Wt. sched 1
Wt. sched 2
Wt. sched 3
Wt. sched 4
Wt. sched 5
Solution Cost (factor over optimal)
1.2
1.1
1.0
1
2
3
4
Wall time relative to serial A*

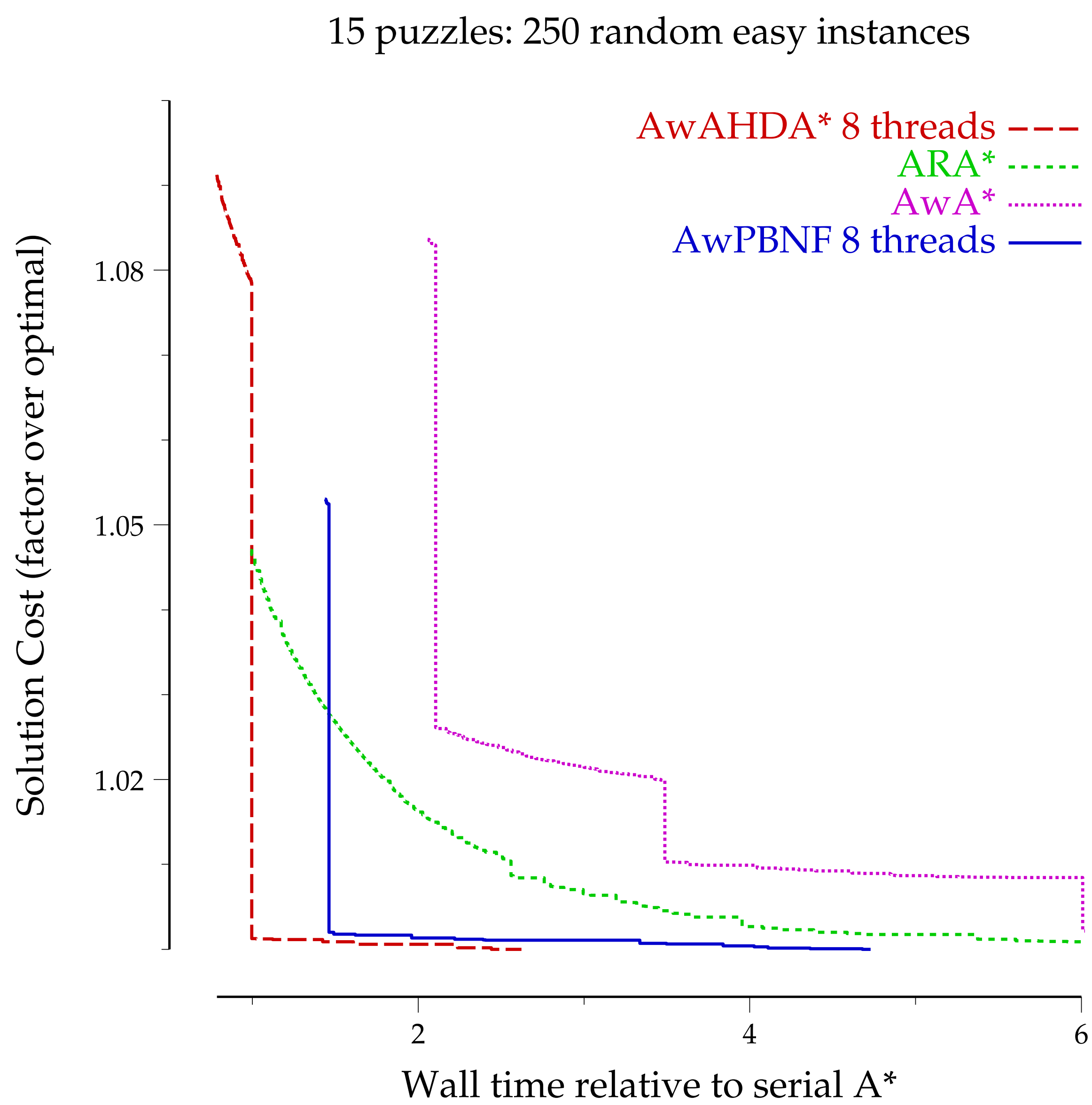
# Four-way Grid Pathfinding 5000x5000





# Easy 15-Puzzles





# Pruning poor nodes



**Theorem:** Can prune a node $n$ if $w \cdot f(n) \geq g(s)$, where $s$ is the incumbent solution and $w$ is the desired bound.

# Pruning duplicate nodes



**Theorem:** No need to re-expand $d$ if the old $g(d) \leq g(n) + w \cdot c^*(n, d)$, where $c^*(n, d)$ is the cost of the path from $n$ to $d$.