Suboptimal and Anytime Heuristic Search on Multi-core Machines

Ethan Burns\textsuperscript{1}, Seth Lemons\textsuperscript{1}, Wheeler Ruml\textsuperscript{1} and Rong Zhou\textsuperscript{2}

\textsuperscript{1}University of New Hampshire

\textsuperscript{2}PARC

[Many thanks to NSF grant IIS-0812141]
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

Introduction

- Overview
- Parallel Search

PRA*
PBNF
Optimal Search
Suboptimal Search
Anytime Search
Conclusion
Introduction

- Overview
- Parallel Search

PRA*
PBNF
Optimal Search
Suboptimal Search
Anytime Search
Conclusion
- 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)
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.

![Diagram of a grid with blue nodes and arrows indicating connections between them.](image)
- Work is divided among threads using a special hash function based on abstraction.
  - \( n \) blocks have an open and closed list.
Work is divided among threads using a special hash function based on abstraction.

- An $n$block and its successors: *duplicate detection scope*. 

---

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

- Introduction
- PRA*
- PBNF
  - Abstraction
  - $N$blocks
  - Detection Scope
    - Disjoint Scopes
    - PBNF
- Optimal Search
- Suboptimal Search
- Anytime Search
- Conclusion

---

Ethan Burns (UNH) Heuristic Search for Multi-core – 9 / 38
Work is divided among threads using a special hash function based on abstraction.

- *Disjoint* duplicate detection scopes searched in parallel.
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)
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.
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.
Grid Unit Four-way 5000x5000

- PRA*
- A*
- HDA*
- APRA*
- AHDA*
- PBNF

wall time (seconds)

threads
Four-way Grid Pathfinding 5000x5000

Grid Unit Four-way 5000x5000

wall time (seconds)

threads

Introduction

PRA*

PBNF

Optimal Search

■ Algorithms

■ Grid Pathfinding

■ Sliding Tiles

■ Planning

■ Summary

Suboptimal Search

Anytime Search

Conclusion

PRA* 

A*

HDA*

APRA*

AHDA*

PBNF

Ethan Burns (UNH) Heuristic Search for Multi-core – 15 / 38
Easy 15-Puzzles

15 puzzles: 250 random easy instances

![Graph showing the performance of different algorithms (A*, HDA*, PRA*, AHDA*, APRA*, PBNF) across varying numbers of threads (2 to 8). The y-axis represents wall time in seconds, and the x-axis represents the number of threads.](image-url)
## Optimal STRIPS Planning

### Introduction

<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
</table>

### PRA*

<table>
<thead>
<tr>
<th>PRA*</th>
</tr>
</thead>
</table>

### PBNF

<table>
<thead>
<tr>
<th>PBNF</th>
</tr>
</thead>
</table>

### Optimal Search

- Algorithms
- Grid Pathfinding
- Sliding Tiles
- Planning
- Summary

### Suboptimal Search

### Anytime Search

### Conclusion

#### Wall time in seconds

<table>
<thead>
<tr>
<th>Threads</th>
<th>logistics-6</th>
<th>blocks-14</th>
<th>gripper-7</th>
<th>satellite-6</th>
<th>elevator-12</th>
<th>freecell-3</th>
<th>depots-7</th>
<th>driverlog-11</th>
<th>gripper-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2.30</td>
<td>5.19</td>
<td>117.78</td>
<td>130.85</td>
<td>335.74</td>
<td>199.06</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>3</td>
<td>1.44</td>
<td>7.37</td>
<td>62.61</td>
<td>95.11</td>
<td>215.19</td>
<td>153.71</td>
<td>319.48</td>
<td>334.28</td>
<td>569.26</td>
</tr>
<tr>
<td>5</td>
<td>0.75</td>
<td>5.30</td>
<td>43.13</td>
<td>42.85</td>
<td>243.24</td>
<td>122.00</td>
<td>138.30</td>
<td>99.37</td>
<td>351.87</td>
</tr>
<tr>
<td>7</td>
<td>1.09</td>
<td>3.26</td>
<td>37.62</td>
<td>67.38</td>
<td>211.45</td>
<td>63.47</td>
<td>67.24</td>
<td>89.73</td>
<td>236.93</td>
</tr>
</tbody>
</table>

| APRA*   |             |           |           |             |             |             |           |               |           |
| 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         |           |

| AHDA*   |             |           |           |             |             |             |           |               |           |
| 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 38.43 | 82.84     | 63.65         |           |
| 7       | 0.62 2.02   | 9.21 13.67| 27.02     | 37.02       | 34.66 31.22 | 51.50       |           |               |           |

Ethan Burns (UNH)
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 $\min_{n \in \text{open}} w \cdot f(n) \geq g(s)$.
  - Two new pruning rules: see paper.

- Suboptimal PBNF
  - Sort $n$ block free-list on $\min_{n \in \text{open}} f'(n)$.
### Speedup over serial wA

- wPBNF gave the best performance at all but 1 thread.
- Lower weight gives more speedup.
Korf’s 100 15-Puzzles

**Introduction**

**PRA**

**PBNF**

**Optimal Search**

**Suboptimal Search**

- New: Suboptimal
- Grid Pathfinding

**Sliding Tiles**

- Planning
- Summary

**Anytime Search**

**Conclusion**

---

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

**Speedup over serial wA**

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

---

Ethan Burns (UNH)

Heuristic Search for Multi-core – 22 / 38
## STRIPS Planning

### Introduction

- PRA*
- PBNF
- Optimal Search
- Suboptimal Search
- New: Suboptimal
- Grid Pathfinding
- Sliding Tiles
- Planning
- Summary

### Anytime Search

### Conclusion

## Speedup over serial wA*

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

---

Ethan Burns (UNH) Heuristic Search for Multi-core – 23 / 38
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
Simple to convert PRA* and PBNF to anytime.

- Sort open lists on $f'(n) = g(n) + w \cdot h(n)$.
- Stop when $\min_{n \in \text{open}} f(n) \geq g(s)$ (same as optimal).

Anytime PBNF

- Sort $n$ block free-list on $\min_{n \in \text{open}} f'(n)$.

Parallel analogue to Anytime Weighted A* (Hansen and Zhou, JAIR 2007)
<table>
<thead>
<tr>
<th></th>
<th>AwAPRA*</th>
<th>AwAHDA*</th>
<th>AwPBNF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>logistics-6</td>
<td>1.09</td>
<td>1.06</td>
<td>1.40</td>
</tr>
<tr>
<td>blocks-14</td>
<td>1.36</td>
<td>7.76</td>
<td>56.41</td>
</tr>
<tr>
<td>gripper-7</td>
<td>0.78</td>
<td>0.77</td>
<td>0.76</td>
</tr>
<tr>
<td>satellite-6</td>
<td>0.77</td>
<td>0.78</td>
<td>0.78</td>
</tr>
<tr>
<td>elevator-12</td>
<td>0.64</td>
<td>0.67</td>
<td>0.69</td>
</tr>
<tr>
<td>freecell-3</td>
<td>1.37</td>
<td>1.43</td>
<td>4.61</td>
</tr>
<tr>
<td>depots-7</td>
<td>1.24</td>
<td>1.30</td>
<td>1.30</td>
</tr>
<tr>
<td>driverlog-11</td>
<td>1.15</td>
<td>1.19</td>
<td>1.11</td>
</tr>
<tr>
<td>gripper-8</td>
<td>0.61</td>
<td>0.62</td>
<td>0.62</td>
</tr>
<tr>
<td>logistics-6</td>
<td>1.45</td>
<td>1.43</td>
<td>1.81</td>
</tr>
<tr>
<td>blocks-14</td>
<td>2.54</td>
<td>15.63</td>
<td>98.52</td>
</tr>
<tr>
<td>gripper-7</td>
<td>1.77</td>
<td>1.68</td>
<td>1.71</td>
</tr>
<tr>
<td>satellite-6</td>
<td>1.22</td>
<td>1.22</td>
<td>1.26</td>
</tr>
<tr>
<td>elevator-12</td>
<td>0.93</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>freecell-3</td>
<td>3.64</td>
<td>3.75</td>
<td>11.59</td>
</tr>
<tr>
<td>depots-7</td>
<td>3.60</td>
<td>3.64</td>
<td>3.65</td>
</tr>
<tr>
<td>driverlog-11</td>
<td>3.04</td>
<td>3.20</td>
<td>3.05</td>
</tr>
<tr>
<td>gripper-8</td>
<td>1.72</td>
<td>1.67</td>
<td>1.70</td>
</tr>
</tbody>
</table>

**Speedup over serial AwA* (to convergence)**
■ Outperforms serial anytime search.
■ AwPBNF gave the best performance on all but three domains.
■ AwAHDA* occasionally gave much better performance.
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

- Problem Difficulty
- Hull Plots
- Grid Pathfinding
- Sliding Tiles
- New: Pruning
Difficulty versus Advantage over wA*

Sliding Tiles wPBNF v.s. wA*

log10(8)

log10(Times faster than wA*)

log10(Nodes expanded by wA*)

wPBNF-1.4
wPBNF-1.7
wPBNF-2.0
wPBNF-3.0
wPBNF-5.0

Problem Difficulty
Hull Plots
Grid Pathfinding
Sliding Tiles
New: Pruning
Hull Plots

Raw Data for ARA*

Solution Cost (factor over optimal)

Wall time relative to serial A*

- Wt. sched 1
- Wt. sched 2
- Wt. sched 3
- Wt. sched 4
- Wt. sched 5
Four-way Grid Pathfinding 5000x5000

Introduction

PRA*
PBNF

Optimal Search

Suboptimal Search

Anytime Search

Conclusion

Additional Slides
- Problem Difficulty
- Hull Plots
- Grid Pathfinding
- Sliding Tiles
- New: Pruning

Solution Cost (factor over optimal)

Wall time relative to serial A*

AwAHDA* 8 threads
AwPBNF 8 threads
AwA*
ARA*
Easy 15-Puzzles

15 puzzles: 250 random easy instances

Solution Cost (factor over optimal)

Wall time relative to serial A*

- AwAHDA* 8 threads
- ARA*
- AwA*
- AwPBNF 8 threads
**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.
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 \( d \).