Parallel Best-First Search: The Role of Abstraction

Ethan Burns\textsuperscript{1}, Sofia Lemons\textsuperscript{1}, Wheeler Rum\textsuperscript{1} and Rong Zhou\textsuperscript{2}

\textsuperscript{1} University of New Hampshire

\textsuperscript{2} PARC, Palo Alto Research Center

[Many thanks to NSF grant IIS-0812141 and the DARPA CSSG program.]
Given:

- Initial state, Goal test, Expand function
- Cost-to-go estimate (heuristic)

Find:

- Cheapest path to a goal state

Some Properties:

- Unknown maximum depth (possibly infinite)
- Possibly duplicate states (graph, not a tree)
- Real valued edge costs
Best-first Search

- \[ f(n) = g(n) + h(n) \]

- \( g \) is the cost accrued from initial state to \( n \)
- \( h \) is the estimated remaining cost to go from \( n \)
- \( f \) is the estimated solution cost under \( n \)

- Search in order of lowest \( f \)
Naive Parallel Search

Introduction
- Heuristic Search
- Best-first Search
- Parallel Search
- PRA*
- PBNF
- Optimal Search
- Suboptimal Search
- Conclusion

Ethan Burns (UNH) Abstraction in Multicore Heuristic Search – 4 / 31
Naive Parallel Search

Introduction

- Heuristic Search
- Best-first Search
- Parallel Search
- PRA*
- PBNF

Optimal Search

Suboptimal Search

Conclusion
Heuristic Search
Best-first Search
Parallel Search
PRA*
PBNF
Optimal Search
Suboptimal Search
Conclusion
Distribute states among threads using a hash function.

- Each state has a home thread.
- Duplicate detection can be performed locally at each thread.
May need to communicate states between threads at each generation.

Non-blocking: HDA* (Kishimoto et al., best paper award ICAPS 2009)
Search space can be divided by abstraction too.

Abstract PRA* (APRA*) and Abstract HDA* (AHDA*)
Search space can be divided by abstraction too.

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

- Threads search groups of states called $n$blocks.
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*.
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.
We have seen:

- Review of heuristic search
- Parallel search algorithms
  - PRA* (HDA*, ARPA*, AHDA*)
  - PBNF

Next:

- Parallel algorithms in optimal search.
- Parallel algorithms in bounded suboptimal search.
Optimal Search
Grid pathfinding:
- Navigate from start to goal in a grid maze
- Lots of ways to get to each state (lots of duplicates)

Sliding piles:
- Slide tiles around from initial to goal configuration
- Few ways to get to each state (few duplicates)

Domain independent planning:
- Find a plan in a domain given in a STRIPS-like language
- Lots of variety
- Poor quality heuristic estimate
Abstraction in PRA*

Introduction

PRA*

PBNF

Optimal Search
- Domains
  - APRA*
- Grid Pathfinding
- Sliding Tiles
- Planning
- Summary

Suboptimal Search

Conclusion

Grid Unit Four-way 5000x5000

wall time (seconds)

threads

PRA* —
A* —-
HDA* —-
Async —

Ethan Burns (UNH)
Abstraction in PRA*

Introduction

PRA*

PBNF

Optimal Search
- Domains
  - APRA*
- Grid Pathfinding
- Sliding Tiles
- Planning
- Summary

Suboptimal Search

Conclusion

Grid Unit Four-way 5000x5000

wall time (seconds)

threads

PRA*  
A*  
HDA*  
APRA*  
AHDA*  
PBNF

Async

Abstraction
Grid Pathfinding

Grid Unit Four-way

Perfect speedup
Achievable speedup
PBNF
AHDA*

speedup over serial A*

threads

8
6
4
2
Perfect speedup
Achievable speedup
PBNF
AHDA*

Ethan Burns (UNH) Abstraction in Multicore Heuristic Search – 18 / 31
Easy 15-Puzzles

15 Puzzles 250 Easy

Perfect speedup
Achievable speedup

PBNF
AHDA*

speedup over serial A*

threads
### Wall times (seconds)

<table>
<thead>
<tr>
<th>threads</th>
<th>A*</th>
<th>AHDA*</th>
<th>PBNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>logistics-6</td>
<td>2.30</td>
<td>0.40</td>
<td>0.62</td>
</tr>
<tr>
<td>blocks-14</td>
<td>5.19</td>
<td>2.13</td>
<td>2.02</td>
</tr>
<tr>
<td>gripper-7</td>
<td>117.78</td>
<td>12.69</td>
<td>9.21</td>
</tr>
<tr>
<td>satellite-6</td>
<td>130.85</td>
<td>18.24</td>
<td>13.67</td>
</tr>
<tr>
<td>elevator-12</td>
<td>335.74</td>
<td>57.10</td>
<td>27.02</td>
</tr>
<tr>
<td>freecell-3</td>
<td>199.06</td>
<td>27.37</td>
<td>37.02</td>
</tr>
<tr>
<td>depots-7</td>
<td>-</td>
<td>39.10</td>
<td>34.66</td>
</tr>
<tr>
<td>driverlog-11</td>
<td>-</td>
<td>48.91</td>
<td>31.22</td>
</tr>
<tr>
<td>gripper-8</td>
<td>-</td>
<td>76.34</td>
<td>51.50</td>
</tr>
</tbody>
</table>

*Note: AHDA* stands for Abstraction Heuristic Directed A*.*
PBNF gave the best performance and scalability across all except two 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
Weighted A* searches on $f' = g + w \cdot h$

- Finds solutions within a factor $w$ of optimal

Converting PRA* and PBNF to bounded suboptimal (wPRA* and wPBNF)

- Sort open lists on $f'(n) = g(n) + w \cdot h(n)$.
- PBNF: Sort $n$ block free-list on $\min_{n \in \text{open}} f'(n)$.

Non-strict $f'$ ordering

- Prove bound: Stop when $\min_{n \in \text{open}} w \cdot f(n) \geq g(s)$.
- Two pruning rules: see paper.
Speedup over serial wA

- wPBNF gave the best performance at all but 1 thread.
- Lower weight gives more speedup.
<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>0.84</td>
<td>1.50</td>
<td>1.90</td>
<td>2.33</td>
<td>2.37</td>
<td>2.39</td>
<td>2.39</td>
<td>2.47</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>1.52</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.
Sliding Tiles wPBNF v.s. wA*

log10(Times faster than wA*)

log10(8)

log10(Nodes expanded by wA*)

wPBNF-1.4 ●
wPBNF-1.7 +
wPBNF-2.0 W
wPBNF-3.0 <
wPBNF-5.0 S
### 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.
Conclusion
Parallel search can make your programs run faster today.

- Multicore is not going away.
- Email me for the code (C++): burns.ethan@gmail.com

PBNF and PRA* are simple and general.

- Easily extendable to suboptimal (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