Best-First Heuristic Search for Multi-Core Machines

Ethan Burns\(^1\), Seth Lemons\(^1\), Rong Zhou\(^2\) and Wheeler Ruml\(^1\)

\(^1\) University of New Hampshire

\(^2\) Palo Alto Research Center

[NSF grant IIS-0812141]
Now we’re into the explicit parallelism multiprocessor era, and this will dominate for the foreseeable future. I don’t see any technology or architectural innovation on the horizon that might be competitive with this approach.

*John Hennessy*

President of Stanford University, Cofounder of MIPS Computer Systems

*(A Conversation with John Hennessy and David Patterson, ACM Queue, December 2006)*
Previous: Parallel Structured Duplicate Detection (Zhou and Hansen, 2007)

- Used abstraction to divide labor.
- Parallelized breadth-first search.
Previous: Parallel Structured Duplicate Detection (Zhou and Hansen, 2007)
- Used abstraction to divide labor.
- Parallelized breadth-first search.

New: Parallel Best $N$Block First Search
- Each thread tries to expand the best nodes.
- Requires care to avoid livelock.
Previous: Parallel Structured Duplicate Detection
### Naive Parallel Search

#### Introduction

#### Previous: PSDD

- Naive Method
- Abstraction
- Detection Scope
- Disjoint Scopes
- PSDD

#### New: PBNF

#### Empirical Evaluation

#### Conclusion
Naive Parallel Search

Introduction

Previous: PSDD

- Naive Method
  - Abstraction
  - Detection Scope
  - Disjoint Scopes
  - PSDD

New: PBNF

Empirical Evaluation

Conclusion
Work is divided among threads using a special hash function based on abstraction.

- Few possible destinations for children.
Work is divided among threads using a special hash function based on abstraction.  

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

**Disjoint duplicate detection scopes** searched in parallel.
- Uses an abstract graph to decompose the search space.
- Uses an abstract graph to decompose the search space.
- Threads proceed breadth-first in parallel.
  - All threads search the same depth layer.
  - All threads synchronize before moving to the next depth.
Parallel Structured Duplicate Detection

- Uses an abstract graph to decompose the search space.
- Threads proceed breadth-first in parallel.
  - All threads search the same depth layer.
  - All threads synchronize before moving to the next depth.
- Heuristic cost-to-go information is used for pruning.
  - Requires an upper-bound or iterative-deepening.
Parallel Structured Duplicate Detection

<table>
<thead>
<tr>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous: PSDD</td>
</tr>
<tr>
<td>- Naive Method</td>
</tr>
<tr>
<td>- Abstraction</td>
</tr>
<tr>
<td>- Detection Scope</td>
</tr>
<tr>
<td>- Disjoint Scopes</td>
</tr>
<tr>
<td>PSDD</td>
</tr>
<tr>
<td>New: PBNF</td>
</tr>
<tr>
<td>Empirical Evaluation</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>

- Uses an abstract graph to decompose the search space.
- Threads proceed breadth-first in parallel.
  - All threads search the same depth layer.
  - All threads synchronize before moving to the next depth.
- Heuristic cost-to-go information is used for pruning.
  - Requires an upper-bound or iterative-deepening.
- Only uses a single lock: when finding free disjoint scopes.
Parallel Structured Duplicate Detection

- Uses an abstract graph to decompose the search space.
- Threads proceed breadth-first in parallel.
  - All threads search the same depth layer.
  - All threads synchronize before moving to the next depth.
- Heuristic cost-to-go information is used for pruning.
  - Requires an upper-bound or iterative-deepening.
- Only uses a single lock: when finding free disjoint scopes.

We want a best-first ordering without layer-based synchronization and one lock.
New: Parallel Best $N$Block First Search
Parallel Best \(N\)Block First

1. Search disjoint \(n\)blocks in parallel.
   - Maintain a heap of free \(n\)blocks.
   - Greedily acquire best free \(n\)block (and its scope).
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 $f(n)$ order.
   - Switch $n$ blocks when a better one becomes free.
   - Perform a minimum amount of work before switching.
   - **Approximates** best-first order.
Parallel Best $N$Block First

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 $f(n)$ order.
   - Switch $n$ blocks when a better one becomes free.
   - Perform a minimum amount of work before switching.
   - **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.
Problem:
Problem:
Problem:
Problem:
Problem:
Problem:
Problem:
Problem:
Problem:

- Issues with parallel execution of processes.
Problem:
Problem:
Problem:
Problem:
Problem:
Problem:
Problem:
Problem:
Problem:
Problem:
**No guarantee that a given \( n \)block will become free.**
- In infinite search spaces, there can be livelock.

**Solution:** check for *hot* \( n \)blocks
- Flag better \( n \)blocks as *hot*
- Release an \( n \)block to free an interfered hot \( n \)block.
Solution:
Solution:
Solution:
Solution:
Solution:
Empirical Evaluation
Empirical Evaluation

Software

- C++
- POSIX threads
- jemalloc (Grids and Tiles) / custom allocator (STRIPS planning)
- Fedora 9

Hardware

- Dual quad-core Intel Xeon E5320 1.86GHz 64-bits
- 16Gb RAM

Domains

- Grid pathfinding
  - Abstraction: coarser grid
- 15-puzzles (easy 43 of Korf’s 100)
  - Abstraction: ignore some tile numbers
- STRIPS planning
  - Abstraction: generated automatically
### Previous Algorithms

**Introduction**

- Previous: PSDD

**New: PBNF**

### Empirical Evaluation

- **Grids**
- **Tiles**
- **Planning**

**Conclusion**

---

**PA**

- 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.

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

- Hash nodes to distribute among processors.
- Synchronized message queues for “incoming” nodes.

**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.
Four-way Grid Pathfinding (Previous Algorithms)

- Introduction
- Previous: PSDD
- New: PBNF

Empirical Evaluation
- Grids
- Tiles
- Planning

Conclusion

Grid Unit 4-Way (Previous Algorithms)

Wall time (seconds)

Threads

- Lock-free PA*
- KBFS
- PA*
- PSDD
- PRA*
- A*
APRA*
- PRA* with a novel abstraction based hashing.
- Limits contention for message queues.

BFPSDD
- PSDD with $f(n)$ layers instead of depth layers.

PBNF
- Acquire the best free $n$ block.

Safe PBNF
- PBNF with livelock prevention.
Four-way Grid Pathfinding (New Algorithms)

Introduction

Previous: PSDD
New: PBNF

Empirical Evaluation
- Grids
- Tiles
- Planning

Conclusion

Ethan Burns (UNH) Heuristic Search for Multi-Core – 18 / 25

Grid Unit 4-Way

wall time (seconds)

threads

Serial A*
APRA*
BFPSDD
SafePBNF
PBNF
Eight-way Grid Pathfinding

Introduction

Previous: PSDD
New: PBNF

Empirical Evaluation

- Grids
- Tiles
- Planning

Conclusion
Easy Sliding 15-Puzzles

Introduction
Previous: PSDD
New: PBNF

Empirical Evaluation
- Grids
- Tiles
- Planning

Conclusion
<table>
<thead>
<tr>
<th></th>
<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>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2.3</td>
<td>5.2</td>
<td>118</td>
<td>131</td>
<td>336</td>
<td>199</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>3</td>
<td>1.5</td>
<td>7.1</td>
<td>60</td>
<td>96</td>
<td>213</td>
<td>150</td>
<td>301</td>
<td>322</td>
<td>528</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.76</td>
<td>5.5</td>
<td>51</td>
<td>49</td>
<td>269</td>
<td>112</td>
<td>144</td>
<td>103</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1.2</td>
<td>3.8</td>
<td>41</td>
<td>66</td>
<td>241</td>
<td>61</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>APRA*</td>
<td></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>0.84</td>
<td>3.7</td>
<td>28</td>
<td>49</td>
<td>169</td>
<td>40</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.2</td>
<td>3.8</td>
<td>41</td>
<td>66</td>
<td>241</td>
<td>61</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>5</td>
<td>0.84</td>
<td>3.7</td>
<td>28</td>
<td>49</td>
<td>169</td>
<td>40</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>7</td>
<td>0.53</td>
<td>2.6</td>
<td><strong>8.6</strong></td>
<td>17</td>
<td><strong>27</strong></td>
<td><strong>36</strong></td>
<td>M</td>
<td>M</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>PNBF</td>
<td></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>1.3</td>
<td>6.3</td>
<td>40</td>
<td>68</td>
<td>157</td>
<td>186</td>
<td>M</td>
<td>M</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.72</td>
<td>3.8</td>
<td>16</td>
<td>34</td>
<td>56</td>
<td>64</td>
<td>M</td>
<td>M</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.58</td>
<td>2.7</td>
<td>11</td>
<td>21</td>
<td>35</td>
<td>44</td>
<td>M</td>
<td>M</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td><strong>0.53</strong></td>
<td><strong>2.6</strong></td>
<td><strong>8.6</strong></td>
<td>17</td>
<td><strong>27</strong></td>
<td><strong>36</strong></td>
<td>M</td>
<td>M</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>SafePBNF</td>
<td></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>1.2</td>
<td>6.2</td>
<td>40</td>
<td>77</td>
<td>150</td>
<td>127</td>
<td>156</td>
<td>154</td>
<td>235</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.64</td>
<td>2.7</td>
<td>17</td>
<td>24</td>
<td>54</td>
<td>47</td>
<td>63</td>
<td>60</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.56</td>
<td>2.2</td>
<td>11</td>
<td>17</td>
<td>34</td>
<td>38</td>
<td>43</td>
<td>39</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.62</td>
<td>2.0</td>
<td>9.2</td>
<td>14</td>
<td>27</td>
<td>37</td>
<td><strong>35</strong></td>
<td><strong>31</strong></td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>BFPSDD</td>
<td></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>2.1</td>
<td>7.8</td>
<td>42</td>
<td>62</td>
<td>152</td>
<td>131</td>
<td>167</td>
<td>152</td>
<td>243</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.1</td>
<td>4.3</td>
<td>18</td>
<td>24</td>
<td>59</td>
<td>57</td>
<td>67</td>
<td>62</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.79</td>
<td>3.9</td>
<td>12</td>
<td>20</td>
<td>41</td>
<td>48</td>
<td>48</td>
<td>43</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.71</td>
<td>3.4</td>
<td>10</td>
<td><strong>14</strong></td>
<td>32</td>
<td>45</td>
<td>43</td>
<td>35</td>
<td>59</td>
<td></td>
</tr>
</tbody>
</table>

Wall time in seconds
Ethan Burns, Seth Lemons, Wheeler Ruml and Rong Zhou, *Suboptimal and Anytime Heuristic Search on Multi-Core Machines*, ICAPS 2009

- Proof of correctness.
- Bounded suboptimal PBNF.
- Anytime PBNF.
Ethan Burns, Seth Lemons, Wheeler Ruml and Rong Zhou, *Suboptimal and Anytime Heuristic Search on Multi-Core Machines*, ICAPS 2009

- Proof of correctness.
- Bounded suboptimal PBNF.
- Anytime PBNF.

**Future Direction**

- External memory PBNF.
New: Parallel Best $N$ Block First.
New: Parallel Best $N$ Block First.

- Fast
  - Beats all other algorithms used for comparison.
New: Parallel Best $N$ Block First.

- Fast
  - Beats all other algorithms used for comparison.
- Scales well
  - Tested out to eight threads.
New: Parallel Best $N$Block First.

- Fast
  - Beats all other algorithms used for comparison.

- Scales well
  - Tested out to eight threads.

- Easy to use
  - Only requires a user-provided abstraction.
New: Parallel Best $N$Block First.

- Fast
  - Beats all other algorithms used for comparison.
- Scales well
  - Tested out to eight threads.
- Easy to use
  - Only requires a user-provided abstraction.
- “Hot $n$blocks”
  - Prevents livelock.
  - Not much overhead.
New: Parallel Best $N$Block First.

- Fast
  - Beats all other algorithms used for comparison.

- Scales well
  - Tested out to eight threads.

- Easy to use
  - Only requires a user-provided abstraction.

- “Hot $n$blocks”
  - Prevents livelock.
  - Not much overhead.

- Source is freely available:
  http://www.cs.unh.edu/~eaburns
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