Robust Bidirectional Search via Heuristic Improvement

Christopher Wilt and Wheeler Ruml

University of New Hampshire

Thanks to Michael Leighton, NSF (grants 0812141 and 1150068) and DARPA (grant N10AP20029)
Bidirectional Search

Bidirectional?

BHPA

Perimeter Search

KKAdd

Incremental KKAdd

Robustness

Conclusion
- Worst case time complexity of a search is $b^d$
- Constructing two search trees with $b^{d/2}$ nodes each is much faster
- $b^d \gg 2 \cdot b^{d/2}$
Bidirectional Search
- Bidirectional?
- BHPA
- Perimeter Search
- KKAdd
- Incremental KKAdd

Robustness

Conclusion
### BHPA (Pohl 1971)

- **Basic bidirectional heuristic search**
- **Searches from start to goal**
- **Searches from goal to start**
- **Terminates when incumbent solution is:**
  1. Better than best forwards partial path
  2. Better than best backwards partial path
- Basic bidirectional heuristic search
- Search from start to goal
- Search from goal to start
- Terminates when incumbent solution is:
  1. Better than best forwards partial path
  2. Better than best backwards partial path
- This algorithm didn’t live up to its $b^{d/2}$ promise
### BHPA (Pohl 1971)

<table>
<thead>
<tr>
<th>Bidirectional Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ Bidirectional?</td>
</tr>
<tr>
<td>■ BHPA</td>
</tr>
<tr>
<td>■ Perimeter Search</td>
</tr>
<tr>
<td>■ KKAdd</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Incremental KKAdd</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ Robustness</td>
</tr>
<tr>
<td>■ Conclusion</td>
</tr>
</tbody>
</table>

- Basic bidirectional heuristic search
- Searches from start to goal
- Searches from goal to start
- Terminates when incumbent solution is:
  1. Better than best forwards partial path
  2. Better than best backwards partial path
- This algorithm didn’t live up to its $b^{d/2}$ promise
- Kainz and Kaindl (1997) showed:
  - BHPA found the optimal solution early
  - Spent the rest of the time proving optimality
Perimeter Search Algorithm

1. Build a perimeter $P$ by doing a limited backwards search
2. Do the forwards search with improved heuristic
   \[ h(n) = \min_{p \in P} (h(n, p) + g_{rev}(p)) \]
Perimeter Search Algorithm

1. Build a perimeter $P$ by doing a limited backwards search
2. Do the forwards search with improved heuristic
   \[ h(n) = \min_{p \in P} (h(n, p) + g_{rev}(p)) \]
Perimeter Search Algorithm

1. Build a perimeter $P$ by doing a limited backwards search
2. Do the forwards search with improved heuristic
   \[ h(n) = \min_{p \in P} (h(n, p) + g_{rev}(p)) \]
Perimeter Search Algorithm

1. Build a perimeter $P$ by doing a limited backwards search
2. Do the forwards search with improved heuristic

$$h(n) = \min_{p \in P} (h(n, p) + g_{rev}(p))$$
The Good:
- Very fast on some domains (e.g. sliding tiles)
Perimeter Search (Dillenburg 1994)

The Good:
- Very fast on some domains (e.g. sliding tiles)

The Bad:
- How much perimeter?
Perimeter Search (Dillenburg 1994)

The Good:
- Very fast on some domains (e.g. sliding tiles)

The Bad:
- How much perimeter?
### Perimeter Search (Dillenburg 1994)

<table>
<thead>
<tr>
<th>Bidirectional Search</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>■ Bidirectional?</td>
<td></td>
</tr>
<tr>
<td>■ BHPA</td>
<td></td>
</tr>
<tr>
<td>■ Perimeter Search</td>
<td></td>
</tr>
<tr>
<td>■ KKAdd</td>
<td></td>
</tr>
</tbody>
</table>

| Incremental KKAdd    |   |
| Robustness           |   |
| Conclusion            |   |

The Good:
- Very fast on some domains (e.g. sliding tiles)

The Bad:
- How much perimeter?
- Using the heuristic $|P|$ times can be slow
Want to get benefit from perimeter without $|P|$ heuristic evaluations

\[ h(p) = \text{Heuristic of } p \]

\[ g_{rev}(p) = \text{Optimal cost of getting from } p \text{ to the goal} \]

\[ \epsilon(p) = g_{rev}(p) - h(p) = \text{Heuristic Error at node } p \]
Want to get benefit from perimeter without $|P|$ heuristic evaluations

$h(p) = \text{Heuristic of } p$

$g_{rev}(p) = \text{Optimal cost of getting from } p \text{ to the goal}$

$\epsilon(p) = g_{rev}(p) - h(p) = \text{Heuristic Error at node } p$

Theorem: If $h$ is consistent, all nodes outside $P$ have at least $\min_{p \in P} (g_{rev}(p) - h(p))$ heuristic error.
KKAdd (Kainz and Kaindl 1997)

A* with KKAdd Heuristic

1. Build a perimeter $P$ by doing a limited backwards search
2. Do the forwards search with improved heuristic

$$h_{KKAdd}(n) = h(n) + \min_{p \in P} (g_{rev}(p) - h(p))$$

$g_{rev}(p) = \text{optimal cost of getting from the goal to } p$

$h(p) = \text{heuristic at node } p$

$g_{rev}(p) - h(p) = \text{error in the heuristic at node } p$
KKAdd (Kainz and Kaindl 1997)

A* with KKAdd Heuristic

1. Build a perimeter $P$ by doing a limited backwards search
2. Do the forwards search with improved heuristic
   \[ h_{KKAdd}(n) = h(n) + \min_{p \in P} (g_{rev}(p) - h(p)) \]

$g_{rev}(p) = \text{optimal cost of getting from the goal to } p$

$h(p) = \text{heuristic at node } p$

$g_{rev}(p) - h(p) = \text{error in the heuristic at node } p$

$\min_{p \in P} (g_{rev}(p) - h(p)) = hAdd$

Called $hAdd$ because we add this quantity to the heuristic
How does KKAdd reduce search effort?

A* expands nodes in $f$ layers
How does KKAdd reduce search effort?

The KKAdd heuristic can reduce the number of $f$ layers. Now $h = h + h_{\text{Add}}$
KKAdd Results

Bidirectional Search
- Bidirectional?
- BHPA
- Perimeter Search
- KKAdd

Incremental KKAdd

Robustness

Conclusion

14 disk Hanoi

Time

A*  KKAdd 10  KKAdd 100  KKAdd 1000  KKAdd 10000  KKAdd 100000  KKAdd 1000000  KKAdd 1500000

Christopher Wilt (UNH) Robust Bidirectional Search via Heuristic Improvement – 10 / 19
KKAdd Results

Bidirectional Search
- Bidirectional?
- BHPA
- Perimeter Search
- KKAdd

Incremental KKAdd

Robustness

Conclusion

Dynamic Robot (200x200)

Time

A*  KKAdd 10  KKAdd 100  KKAdd 1000  KKAdd 10000  KKAdd 100000  KKAdd 1000000
KKAdd heuristic can speed up finding optimal solutions

How much backwards search?

Too little backwards search
   - No benefit

Too much backwards search
   - Backwards search takes more time than plain A*
Incremental KKAdd
■ With KKAdd, all the backwards search is done up front
With KKAdd, all the backwards search is done up front.

With minor modifications, it is possible to alternate forwards and backwards searches.
With KKAdd, all the backwards search is done up front

With minor modifications, it is possible to alternate forwards and backwards searches

Specify perimeter size as a proportion of all expansions
Adjusts backwards expansions to **problem** and **instance** difficulty
Proportion Advantages

- Adjusts backwards expansions to problem and instance difficulty
- Worst case bound:
  - \( h_{Add} = 0 \) so forwards search is the same as A*
  - Backwards work \( < 2 \cdot A^* \) work \( \cdot proportion \)
  - e.g. guaranteed to be comparable to A*
<table>
<thead>
<tr>
<th>Bidirectional Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental KKAdd</td>
</tr>
<tr>
<td>- Changing Search Direction Online</td>
</tr>
<tr>
<td>- Proportion Advantages</td>
</tr>
<tr>
<td>Adaptive KKAdd</td>
</tr>
<tr>
<td>- Results</td>
</tr>
<tr>
<td>Robustness</td>
</tr>
<tr>
<td>Conclusion</td>
</tr>
</tbody>
</table>

- No proportion parameter
- Uses information at runtime to select direction to expand
- See paper for more details
Results

Bidirectional Search
- Incremental KKAdd
  - Changing Search Direction Online
  - Proportion
  - Advantages
- Adaptive KKAdd
- Results

Robustness

Conclusion

14 disk Hanoi

Time

A*  KKAdd 100000  Adp Inc KKAdd

0  10  20
Adaptive KKAdd is able to capture the potential of KKAdd without the risk of doing too many or too few backwards expansions.
Robustness
Other bidirectional search algorithms are brittle.
Other bidirectional search algorithms are brittle.

- **Perimeter search (Dillenburg 94)**
  - Fastest algorithm on tiles
  - Sometimes than twice as slow as A*
Other bidirectional search algorithms are brittle.

- **Perimeter search (Dillenburg 94)**
  - Fastest algorithm on tiles
  - Sometimes twice as slow as A*

- **Single Frontier Bidirectional Search (Felner et. al. 2010)**
  - Fastest algorithm on pancake
  - Fails (out of memory or time) on 5 domains.
Other bidirectional search algorithms are brittle.

- Perimeter search (Dillenburg 94)
  - Fastest algorithm on tiles
  - Sometimes than twice as slow as A*

- Single Frontier Bidirectional Search (Felner et. al. 2010)
  - Fastest algorithm on pancake
  - Fails (out of memory or time) on 5 domains.

- KKAdd (Kainz and Kaindl 1997)
  - Requires unforgiving parameter
Other bidirectional search algorithms are brittle.

- Perimeter search (Dillenburg 94)
  - Fastest algorithm on tiles
  - Sometimes than twice as slow as A*

- Single Frontier Bidirectional Search (Felner et. al. 2010)
  - Fastest algorithm on pancake
  - Fails (out of memory or time) on 5 domains.

- KKAdd (Kainz and Kaindl 1997)
  - Requires unforgiving parameter

- Incremental KKAdd
  - Worse case performance comparable to A*
Conclusion
Resuscitated the KKAdd technique

- Previously overlooked
Conclusion: A Robust Bidirectional Search

- Resuscitated the KKAdd technique
  - Previously overlooked
- Improved applicability KKAdd
  - Proportion bounds overhead
  - Adaptive eliminates parameter
Conclusion: A Robust Bidirectional Search

- Resuscitated the KKAdd technique
  - Previously overlooked
- Improved applicability KKAdd
  - Proportion bounds overhead
  - Adaptive eliminates parameter
- Demonstrated the effectiveness of the technique on a number of benchmark domains
  - Up to 7x speedup
  - Never worse than 1.28x worse than A*
Resuscitated the KKAdd technique
  ◆ Previously overlooked

Improved applicability KKAdd
  ◆ Proportion bounds overhead
  ◆ Adaptive eliminates parameter

Demonstrated the effectiveness of the technique on a number of benchmark domains
  ◆ Up to 7x speedup
  ◆ Never worse than 1.28x worse than A*

A framework for robust bidirectional search