Robust Bidirectional Search via Heuristic Improvement

Christopher Wilt and Wheeler Ruml

University of New Hampshire

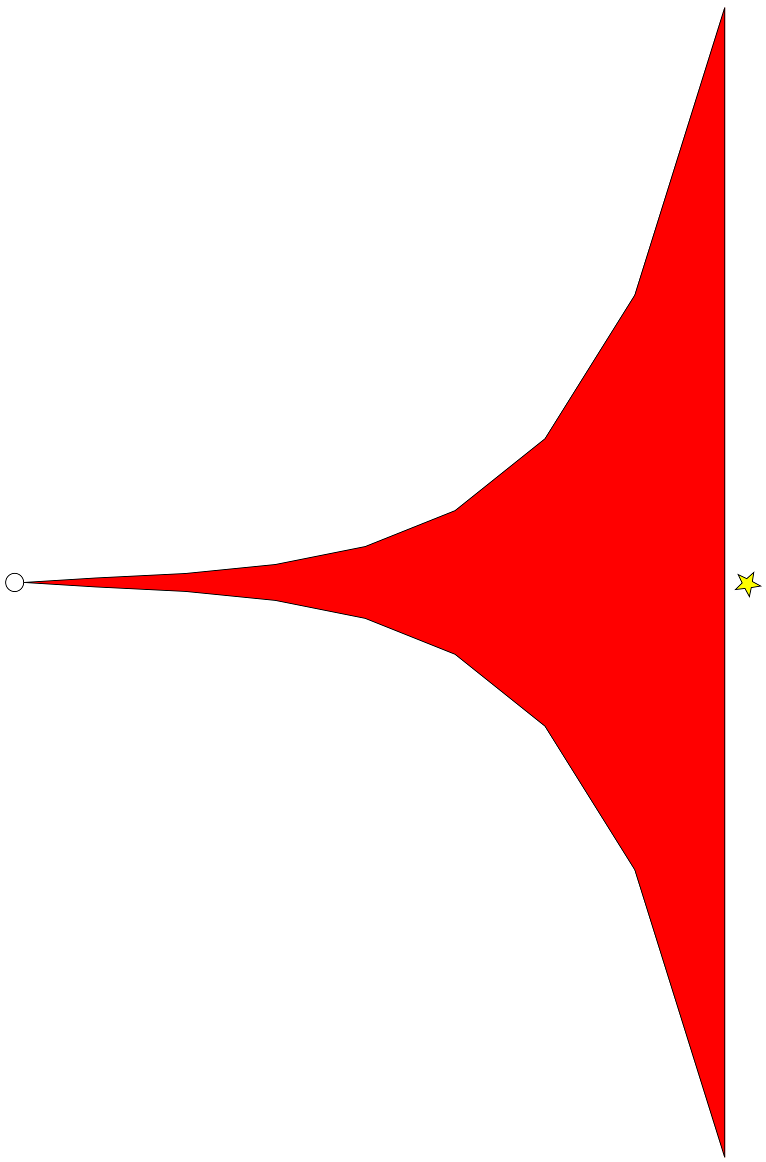
Thanks to Michael Leighton, NSF (grants 0812141 and 1150068) and DARPA (grant N10AP20029)

Bidirectional Search

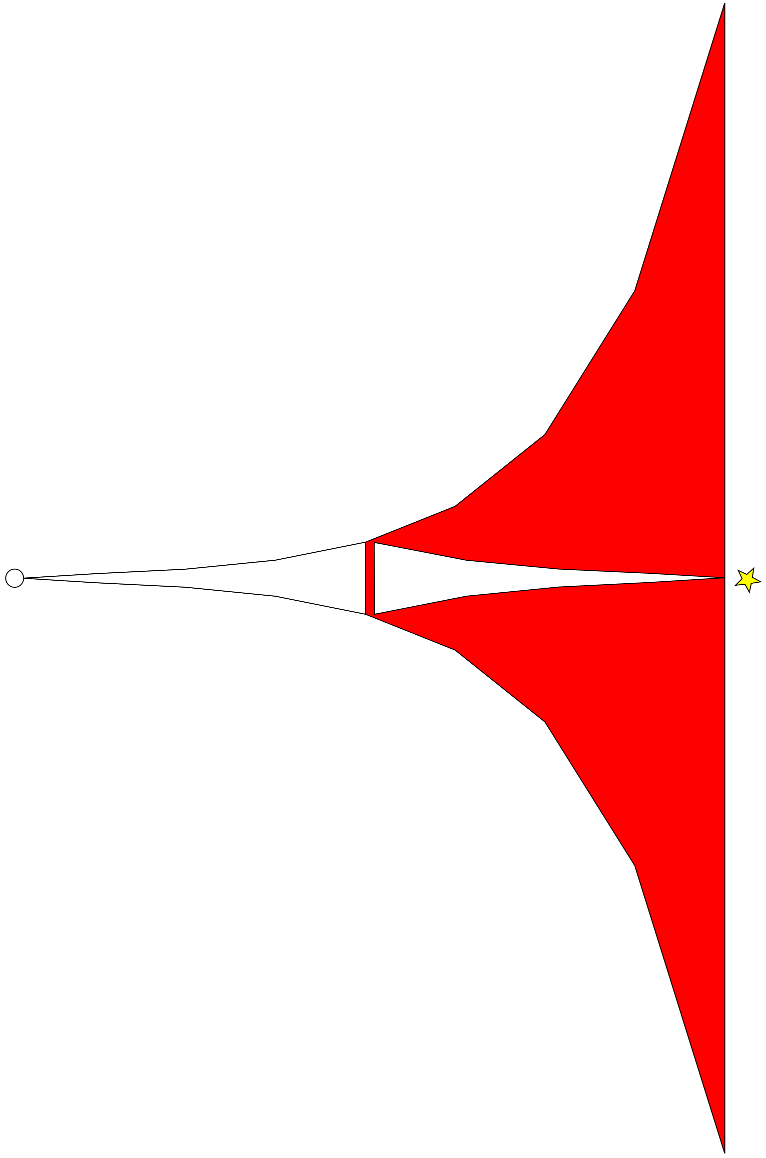
Bidirectional?

- Worst case time complexity of a search is b^d
- Constructing two search trees with $b^{\frac{d}{2}}$ nodes each is much faster
- $b^d \ggg 2 \cdot b^{\frac{d}{2}}$

Bidirectional?



Bidirectional?



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

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
- This algorithm didn't live up to its $b^{\frac{d}{2}}$ promise

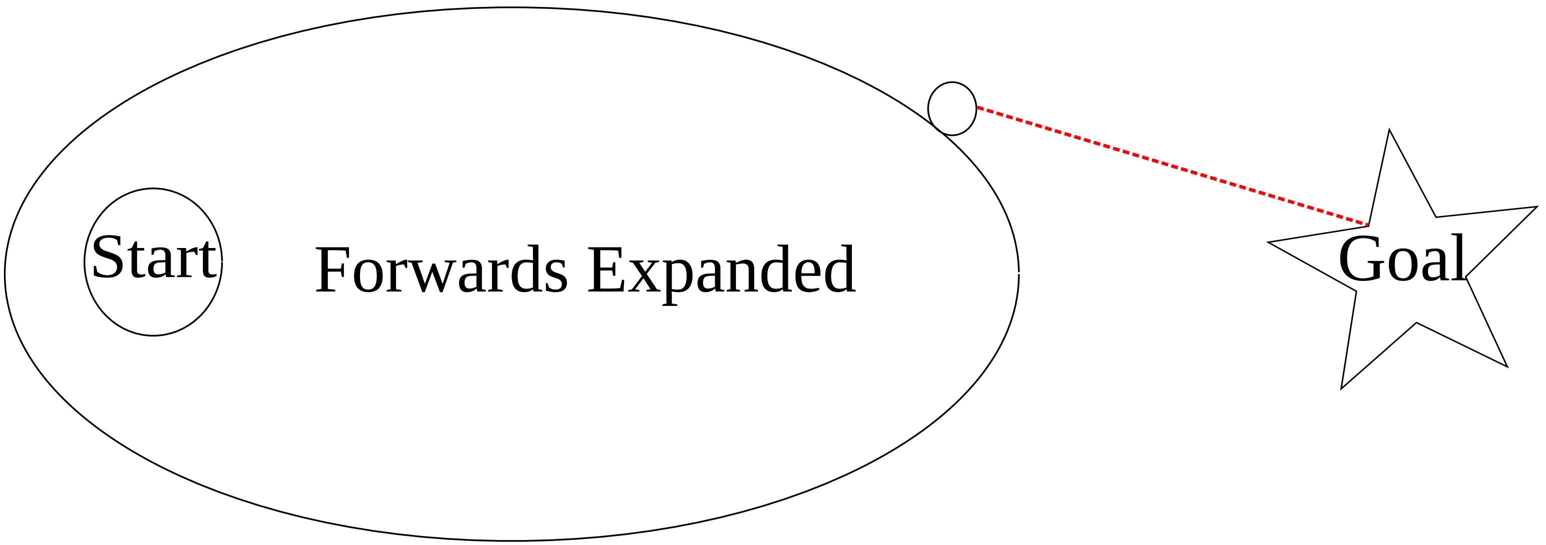
BHPA (Pohl 1971)

- 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
- This algorithm didn't live up to its $b^{\frac{d}{2}}$ promise
- Kainz and Kaindl (1997) showed:
  - BHPA found the optimal solution early
  - Spent the rest of the time proving optimality

Perimeter Search (Dillenburg 1994)

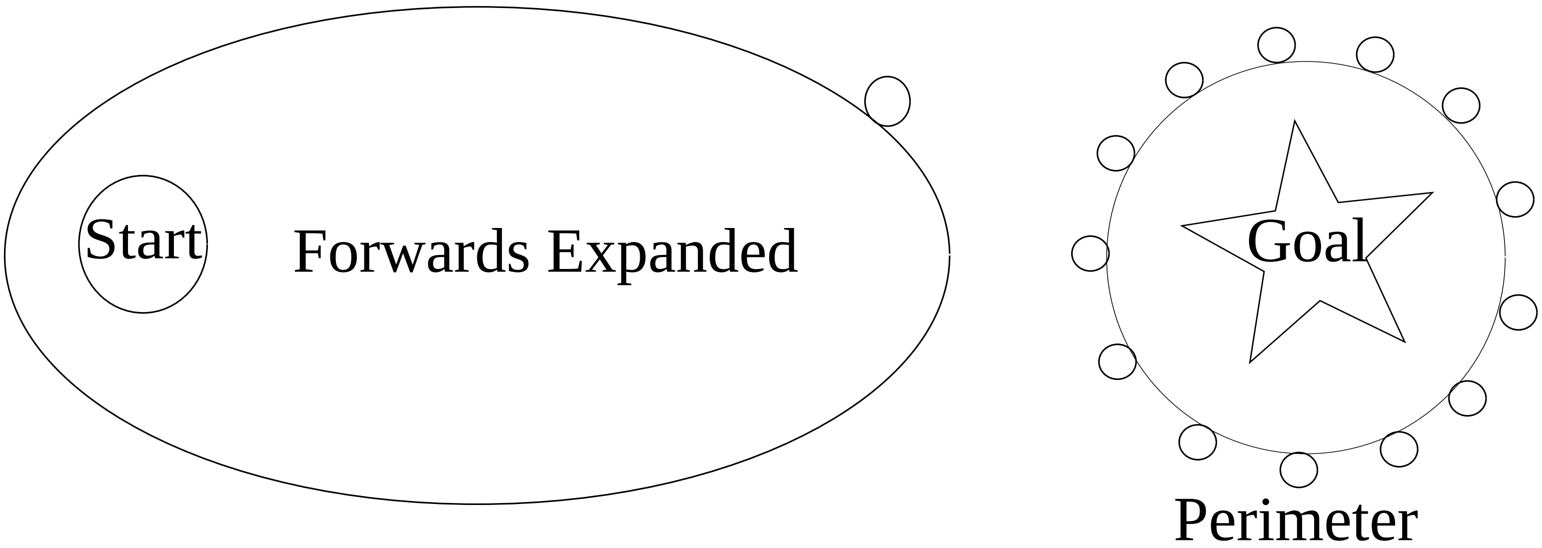
Perimeter Search Algorithm

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



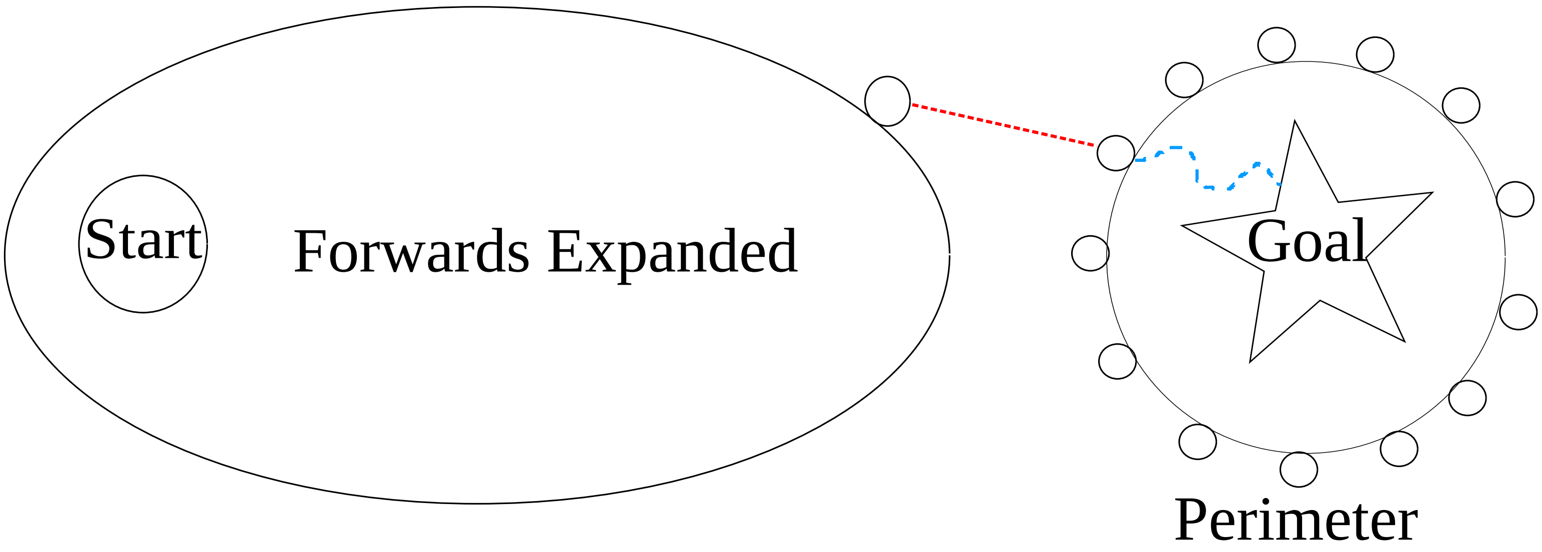
Perimeter Search Algorithm

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



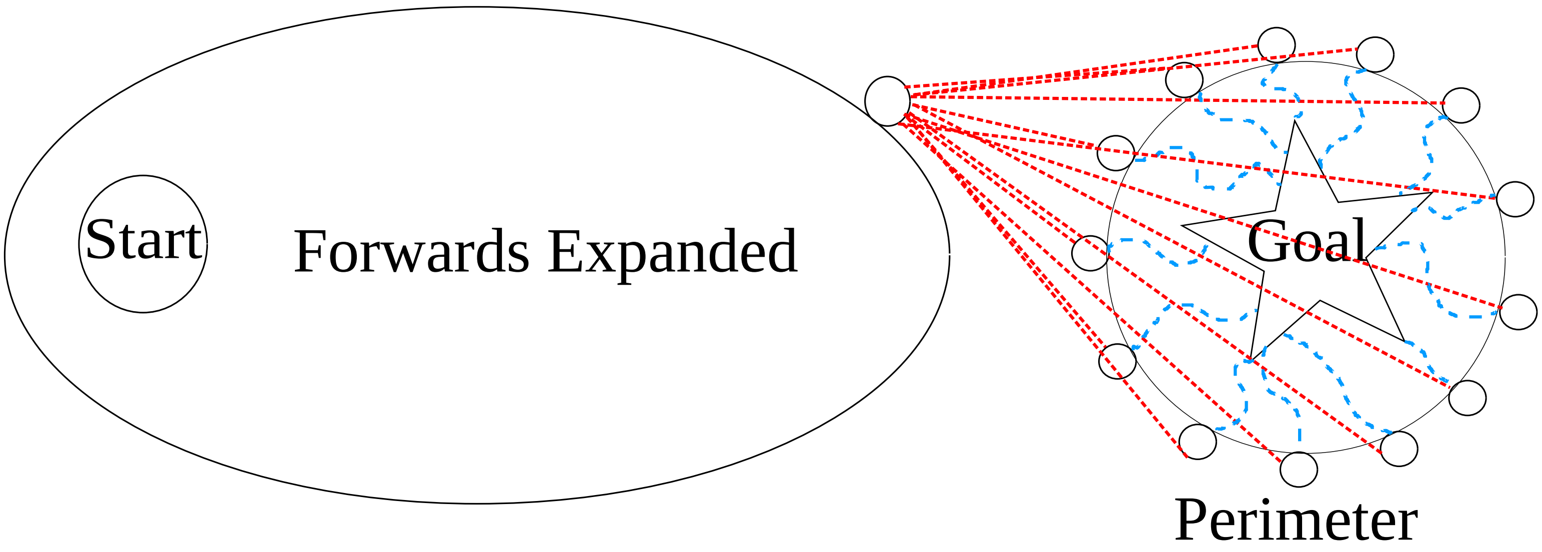
Perimeter Search Algorithm

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



Perimeter Search (Dillenburg 1994)

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?
  - Analyzed by Linares-López (2002)

Perimeter Search (Dillenburg 1994)

The Good:

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

The Bad:

- How much perimeter?
  - Analyzed by Linares-López (2002)
- Using the heuristic $|P|$ times can be slow

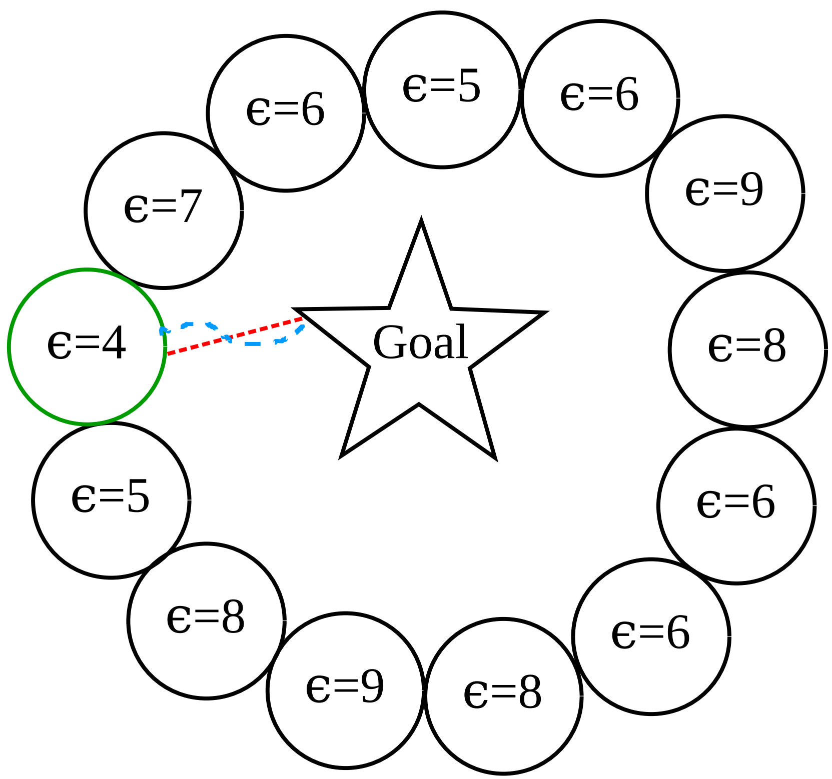
KKAdd (Kainz and Kaindl 1997)

Want to get benefit from perimeter without $|P|$ heuristic evaluations

$h(p)$ = Heuristic of p

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

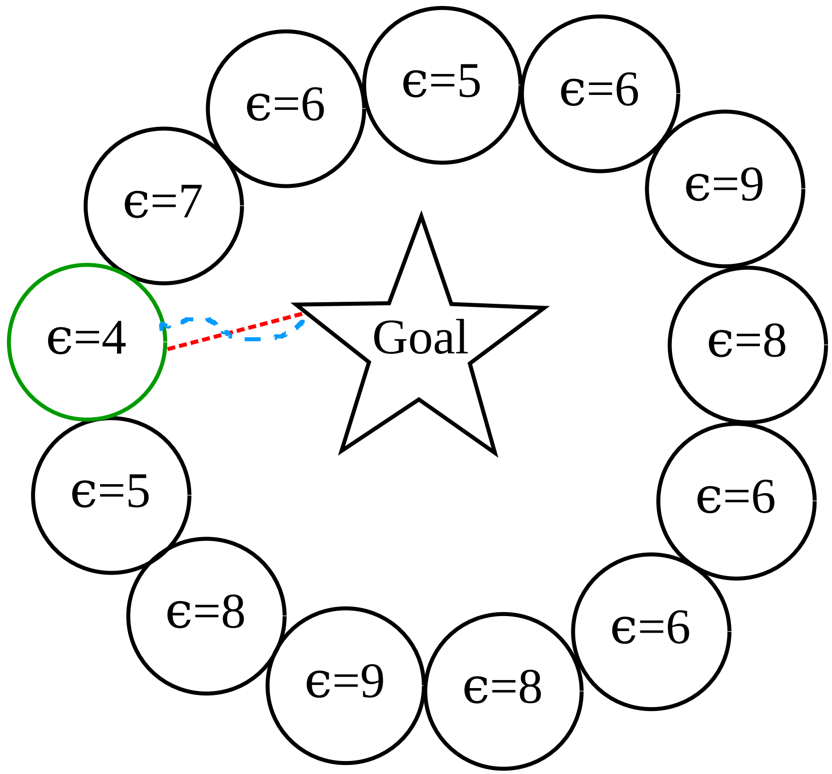
$\epsilon(p) = g_{rev}(p)$ - $h(p)$ = Heuristic Error at node p



KKAdd (Kainz and Kaindl 1997)

Want to get benefit from perimeter without $|P|$ heuristic evaluations

$h(p)$ = Heuristic of p
$g_{rev}(p)$ = Optimal cost of getting from p to the goal
$\epsilon(p) = g_{rev}(p)$ - $h(p)$ = 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.

A* with KKAdd Heuristic

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

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

$h(p) =$ heuristic at node p

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

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

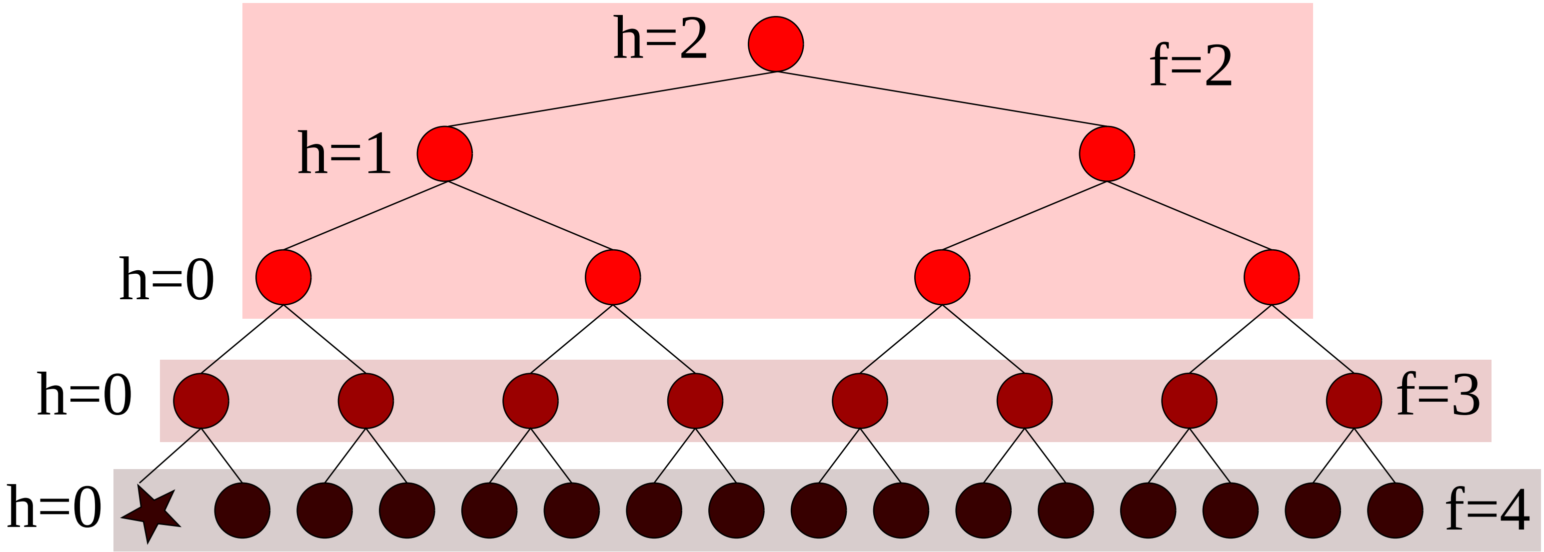
$h(p)$ = heuristic at node p

$g_{rev}(p) - h(p)$ = 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

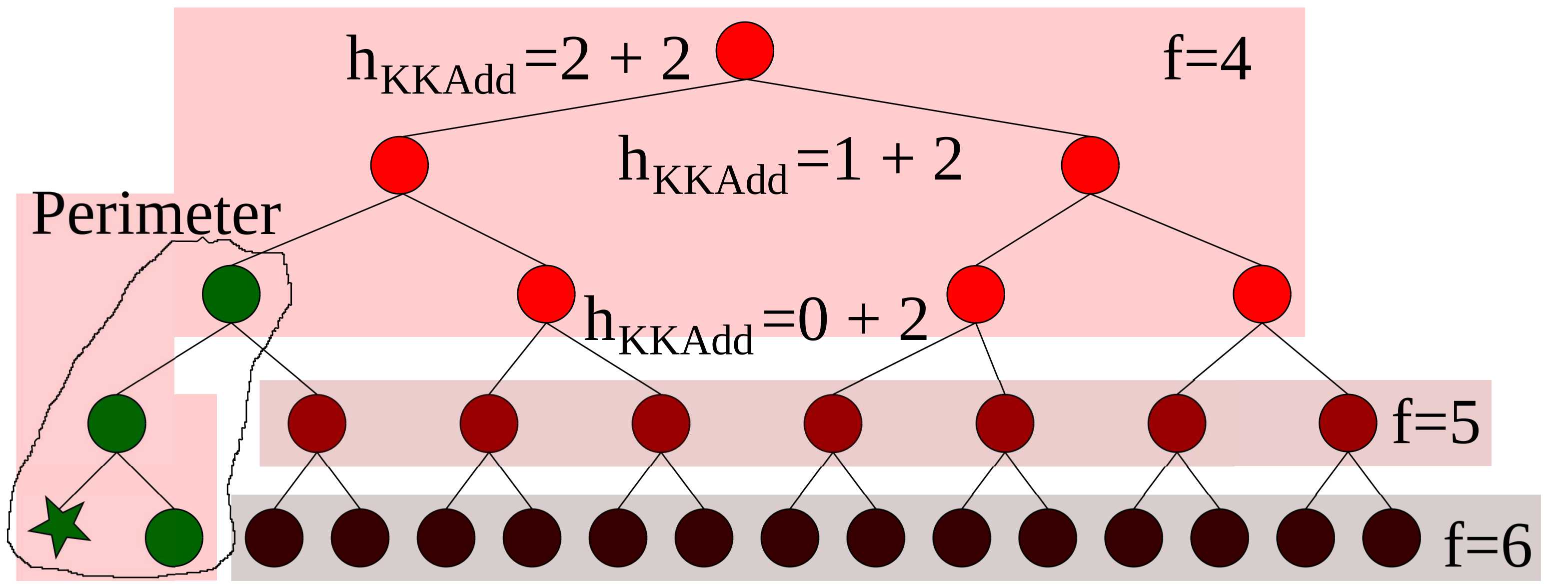
A* expands nodes in f layers



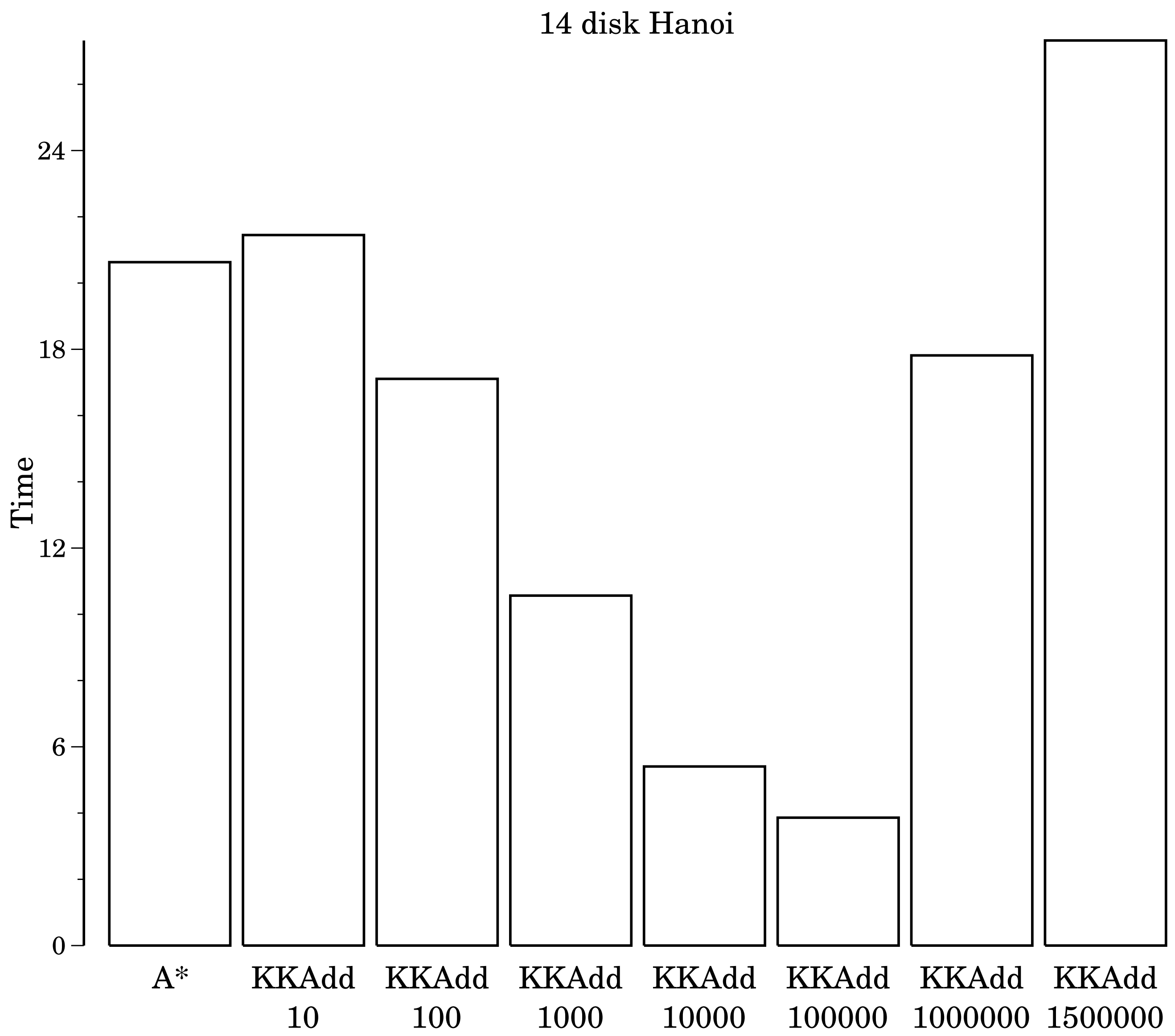
How does KKAdd reduce search effort?

The KKAdd heuristic can reduce the number of f layers

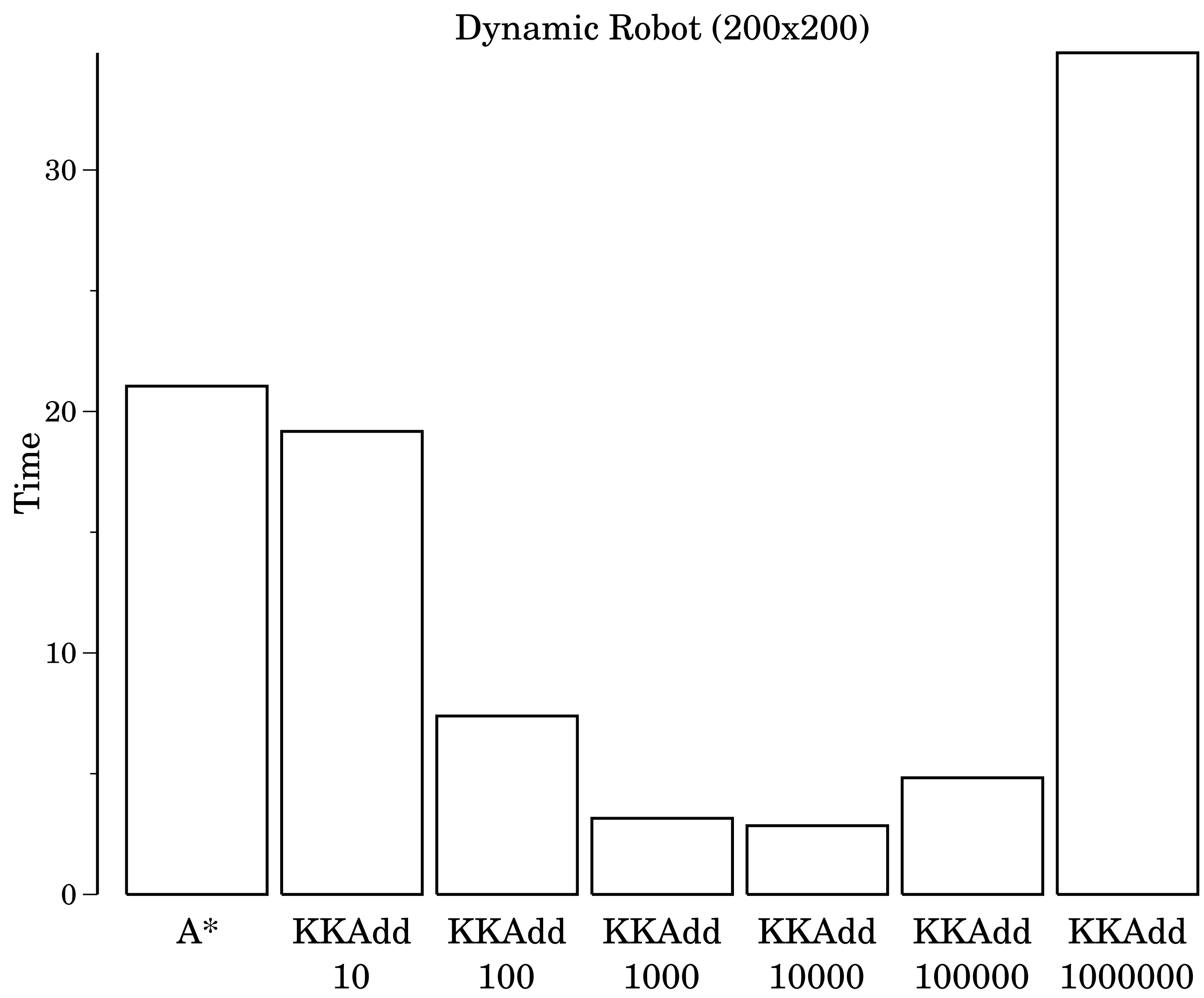
Now $h = h + hAdd$



KKAdd Results



KKAdd Results



KKAdd Results

- 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

Changing Search Direction Online

- 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**

Changing Search Direction Online

- 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

Proportion Advantages

- Adjusts backwards expansions to **problem** and **instance** difficulty

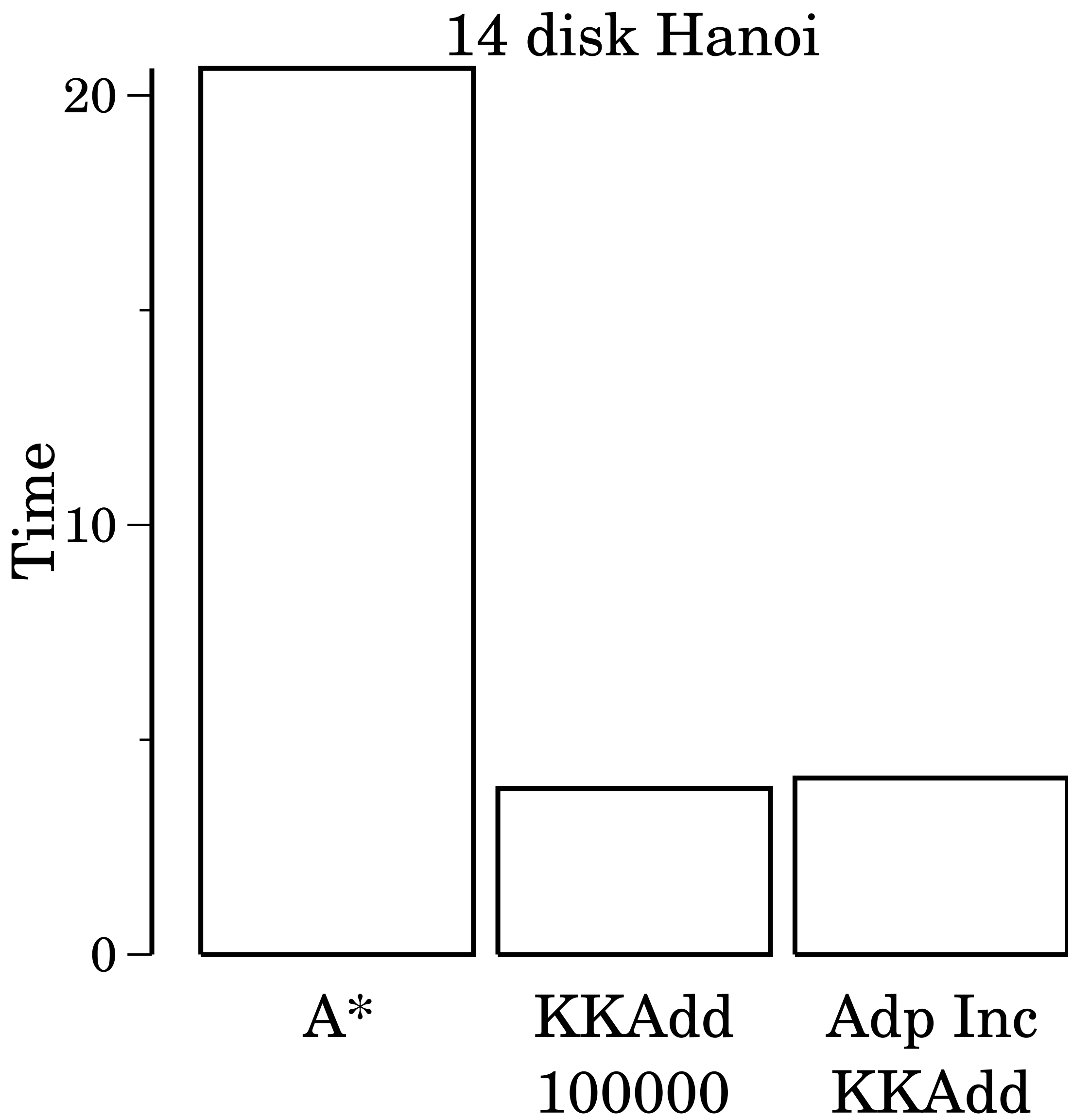
Proportion Advantages

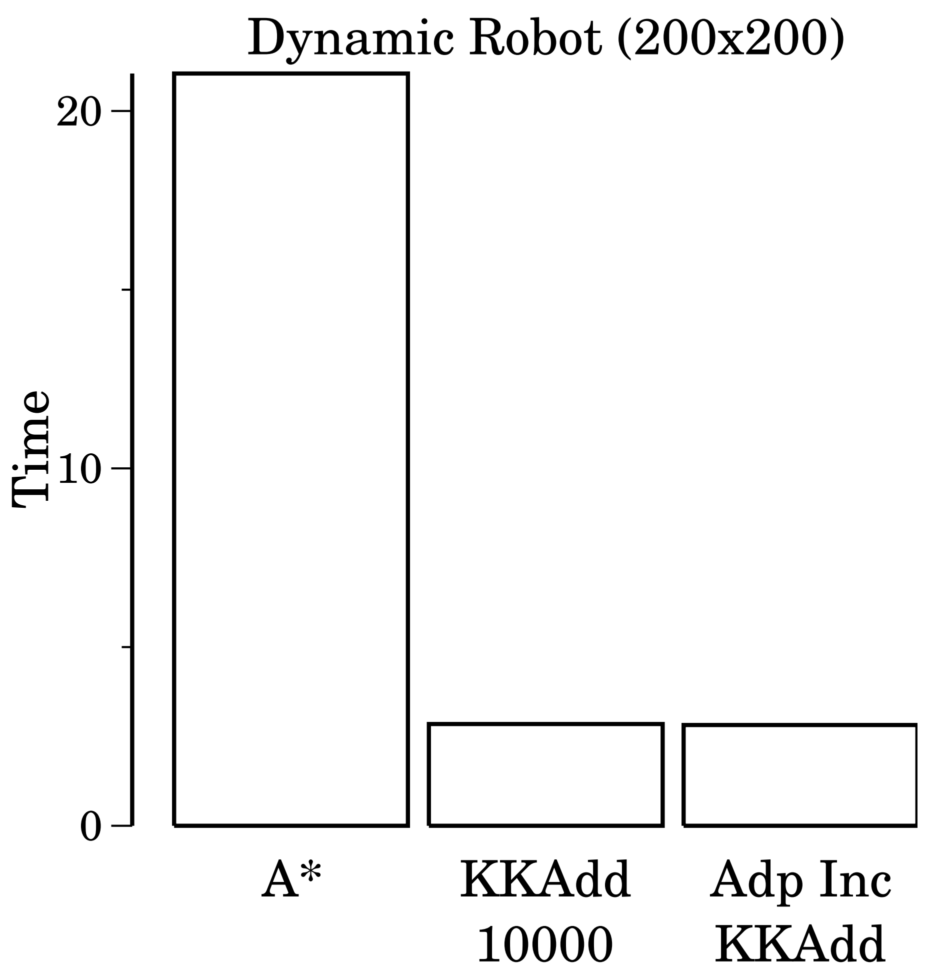
- Adjusts backwards expansions to **problem** and **instance** difficulty
- Worst case bound:
  - $hAdd = 0$ so forwards search is the same as A*
  - Backwards work $< 2\cdot$ A* work $\cdot proportion$
  - e.g. guaranteed to be comparable to A*

Managing Backwards Expansions Online

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

Results





Adaptive KKAdd is able to capture the potential of KKAdd without the risk of doing too many or to few backwards expansions.

Robustness

Bidirectional Brittleness

Other bidirectional search algorithms are brittle.

Bidirectional Brittleness

Other bidirectional search algorithms are brittle.

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

Bidirectional Brittleness

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.

Bidirectional Brittleness

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

Bidirectional Brittleness

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

Conclusion: A Robust Bidirectional Search

- 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*

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*
- **A framework for robust bidirectional search**