Solving the Longest Simple Path Problem with Heuristic Search

YOSSI COHEN

PROF. ARIEL FELNER, DR. RONI STERN
LSP - Definition

- Find the longest *simple* path from Start to Target \((s \rightarrow t)\).
- NP Hard problem - even hard to estimate by a constant factor
Motivation

Real world application:
- VLSI design (integrated circuit design)
- Gray code - error correction
- Robot patrolling
Maximum vs. Minimum

- Unlike many problems that are solved by finding a solution with *minimal cost*, LSP solution requires *maximal reward*.

- Simple Path from 1 to 2
- **MIN** vs **MAX**
- Shortest Path
- **Longest Simple Path (LPP)**
What’s the problem? Let's just use Best First Search!

- [s:0]
- [a:1, b:2]
- [a:1, g:3]
Search Space is Bigger

Both nodes are same place - but they not the same valid operators:

\[[S,W]\] \neq \[[N,S]\]

Moreover one is part of MAX solution and the other is not. We must keep the entire path in every search node.

Due to this the search space is much bigger.
A* and DFBnB Adaptations

- \( f(\cdot) = g(\cdot) + h(\cdot) \)
- A*
  - MAX problems: the first state in the open list is the \textit{maximal} state. (denote \( \text{max}_f \))
  - Stops when open-list is empty or when:
    \[ \text{max}_f \leq \text{BestGoalFound} \]
- DFBnB
  - If \( F < \) best candidate – prune
- Better heuristics are heuristics that tightly \textit{upper} bounds the remaining path
Heuristics

Shortest path heuristics just won't work here
Existing heuristics

Reachable

Biconnected components

Heuristics
Alternate Steps

Works on bipartite graph.
4 connected grid for instance.

Count the groups separately, $|\Delta| \leq 1$

<table>
<thead>
<tr>
<th>S</th>
<th></th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td></td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>→</th>
<th>→</th>
<th>↓</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
<td>←</td>
<td>←</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td></td>
<td>#</td>
<td>←</td>
<td>↓</td>
</tr>
<tr>
<td>#</td>
<td>←</td>
<td>#</td>
<td>←</td>
<td>←</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>↑</td>
<td>↑</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>→</th>
<th>→</th>
<th>↓</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
<td>←</td>
<td>←</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td></td>
<td>#</td>
<td>←</td>
<td>↓</td>
</tr>
<tr>
<td>#</td>
<td>←</td>
<td>#</td>
<td>←</td>
<td>←</td>
</tr>
<tr>
<td></td>
<td>←</td>
<td>↑</td>
<td>↑</td>
<td>T</td>
</tr>
</tbody>
</table>
Run Alternate step (Alt.) on each block of the BCT separately.

\[ h(\text{BCC alt.}) = 33 \]

\[ h(\text{BCC s. alt.}) = 31 \]
How to prune nodes during the search and still guarantee to find the longest simple path?
Search Tree Example

Pruning
When using heuristic search to solve LSP, A* traverse over many similar states.
Basic Symmetry Detection

Same frontier location & same path coverage

On open list: Generated node:

- How to efficiently compare states?
Reachable Dominance Detection

Same frontier & contained (subset or equal) reachable coverage

On open list: Generated node:

- Can prune retroactively!
- How to efficiently compare states?
Reachable Dominance Detection

Pruning Conditions:
1. $N.\text{head} = N'.\text{head}$
2. $|N.\pi| \geq |N'.\pi|$
3. $N'.R \subseteq N.R$
Experimental Results
Experimental Results

- Grid maps with random blocked cells - 360 maps with variety of blocked percentile
- 10 Minutes, solved by all permutations

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Expanded</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A*</td>
<td>DFBnB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NP</td>
<td>BSD</td>
<td>RDD</td>
<td>NP</td>
<td>BSD</td>
<td>RDD</td>
<td>NP</td>
</tr>
<tr>
<td>R</td>
<td>45,211</td>
<td>21,815</td>
<td>16,494</td>
<td>49,772</td>
<td>24,823</td>
<td>10,808</td>
<td>6,155</td>
</tr>
<tr>
<td>R+ALT</td>
<td>34,271</td>
<td>15,708</td>
<td>14,051</td>
<td>38,100</td>
<td>18,286</td>
<td>6,155</td>
<td>1417</td>
</tr>
<tr>
<td>BCC</td>
<td>8,366</td>
<td>2,703</td>
<td>2,271</td>
<td>9,623</td>
<td>3,447</td>
<td>377</td>
<td>110</td>
</tr>
<tr>
<td>BCC+ALT</td>
<td>7,491</td>
<td>2,187</td>
<td>2,077</td>
<td>8,651</td>
<td>2,869</td>
<td>300</td>
<td>82</td>
</tr>
<tr>
<td>BCC+S+ALT</td>
<td>7,348</td>
<td><strong>2,097</strong></td>
<td><strong>2,025</strong></td>
<td>8,499</td>
<td>2,771</td>
<td>311</td>
<td><strong>86</strong></td>
</tr>
</tbody>
</table>

| Grids with random obstacles |               |               |               |               |               |               |               |

RDD prune more but it cost time, Huristic and pruning order, DFBnB is faster.
Experimental Results

- Grid maps with rooms - 400 maps with variety of number of rooms, room size and blocked percentile
- 10 Minutes, solved by all permutations

### A*

- Heuristic and pruning order

### DFBnB

### Runtime

(BCC Based only)
Experimental Results

Success rate – Bigger grids and non-uniform reward 1Hr.

Uniform reward

Life Grid (non-uniform)

DFBnB solve more than A*
Conclusion

Contributions
- Novel heuristic for the longest simple path problem
- Several state space pruning techniques

Results
- All proposed pruning techniques reduce # searched nodes.
- Pruning effectiveness: \( \text{None} \leq \text{BSD} \leq \text{RDP} \)
- Heuristic effectiveness: \( \text{R} \leq \text{R+ALT} \leq \text{BCC} \leq \text{BCC+ALT} \leq \text{BCC+Sep. ALT} \)

Challenge for Future Work
- BSD has the fastest runtime and RDD has the strongest pruning ability
- Challenge: How to get RDD faster?
Thanks!

QUESTIONS?

https://github.com/YossiCohen/Heuristic-Search-Max