Faculty of Engineering Sciences
Department of Software and Information Systems Engineering

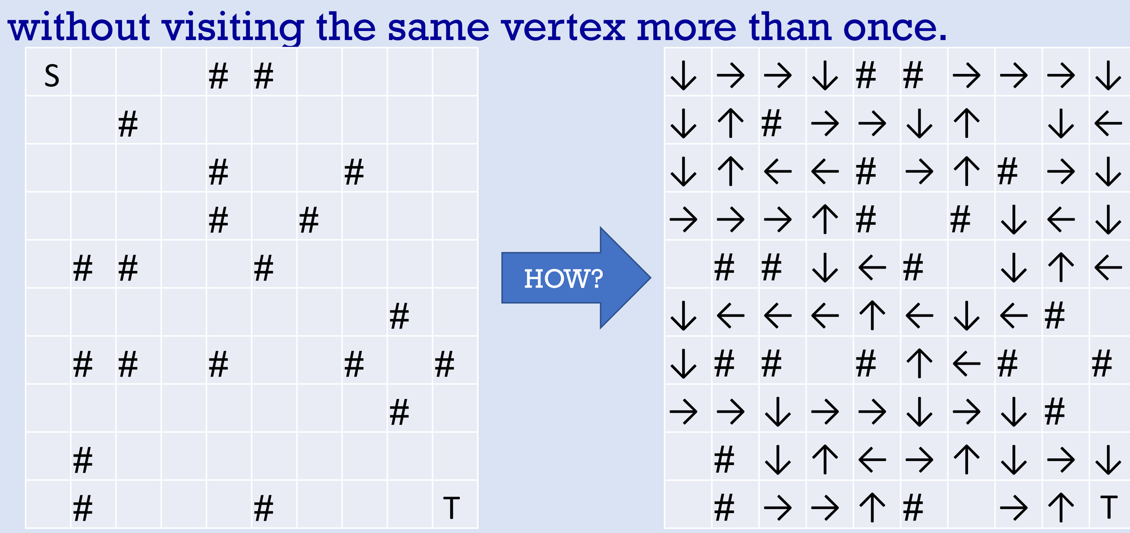
אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Solving the longest simple path problem with heuristic search

Yossi Cohen
Advisors: Prof. Ariel Felner, Dr. Roni Stern

The Problem

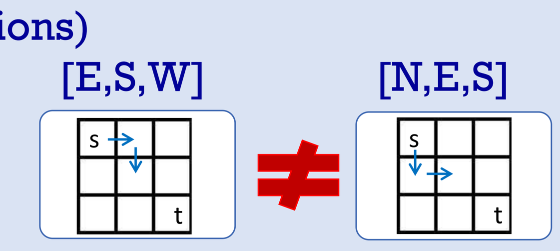
Finding the longest path in a graph from start to goal without visiting the same vertex more than once.



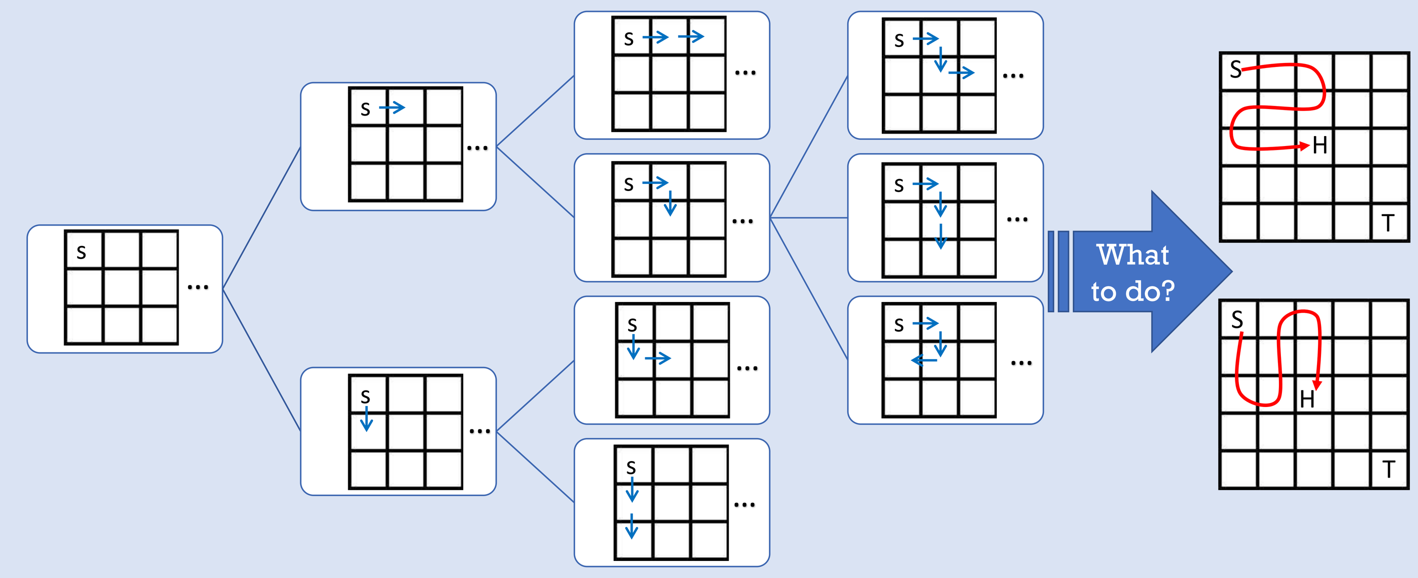
An NP-Hard Problem with applications in VLSI design, error correction code, and robot patrolling.

The Longest Simple Path

- Consider the next two nodes, they reached the same place in the grid but they not share the same valid operators (cardinal directions)



- In order to keep the path *simple* we must keep the entire *path* from the start to the head in every search node – without the full path we can't know what operators are valid.
- that means we will expand for every vertex in the problem graph all the possible paths to it in the search graph - that's A LOT!



Our approach

- prune dominating paths while maintaining optimality
- propose effective admissible heuristics.

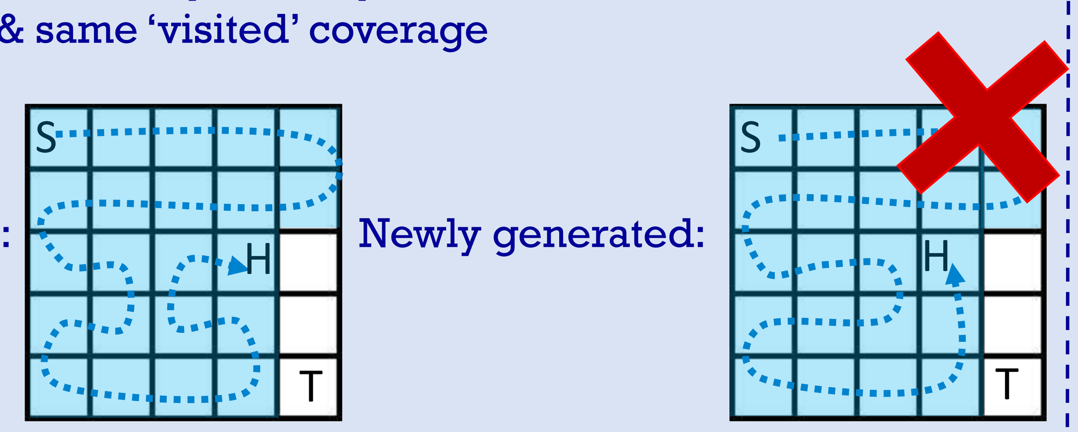
Pruning

How to prune paths during the search and still guarantee to find the longest simple path?

Any time a search node is expand check that a better/equivalent node isn't already exists and maintain the open-list accordingly by removing inferior/equally good search nodes. The down side is that new node should be compared to all existing nodes.

Basic Symmetry Detection

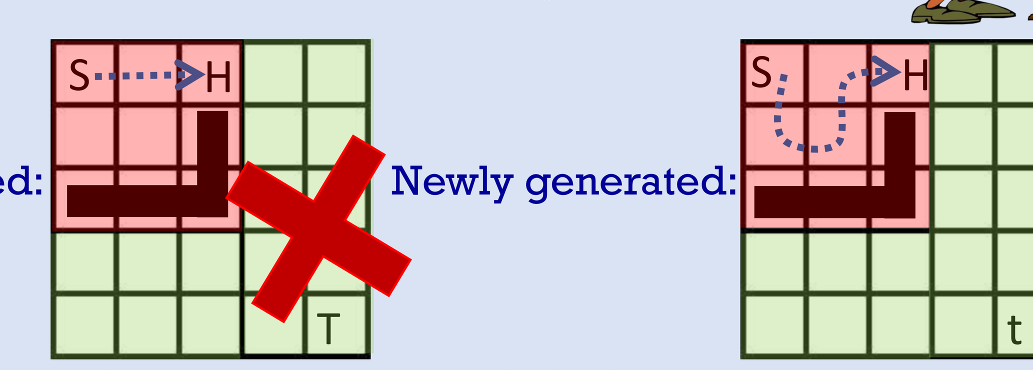
same 'head' location & same 'visited' coverage



Hashing the visited area can help compare nodes efficiently.

Reachable Dominance Property

same 'head' location & same 'reachable' coverage (or sub set of the reachable)



Can prune retrospectively – replace an inferior node with dominated node

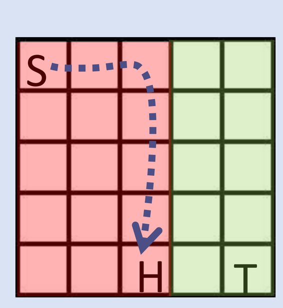
Heuristics

Requirements

- A* and other optimal algorithms require an *admissible* heuristic
- In LSP, an admissible heuristic must *upper bound* the length of the longest path to the goal
- We cant just say h=∞, smaller values means better heuristics!

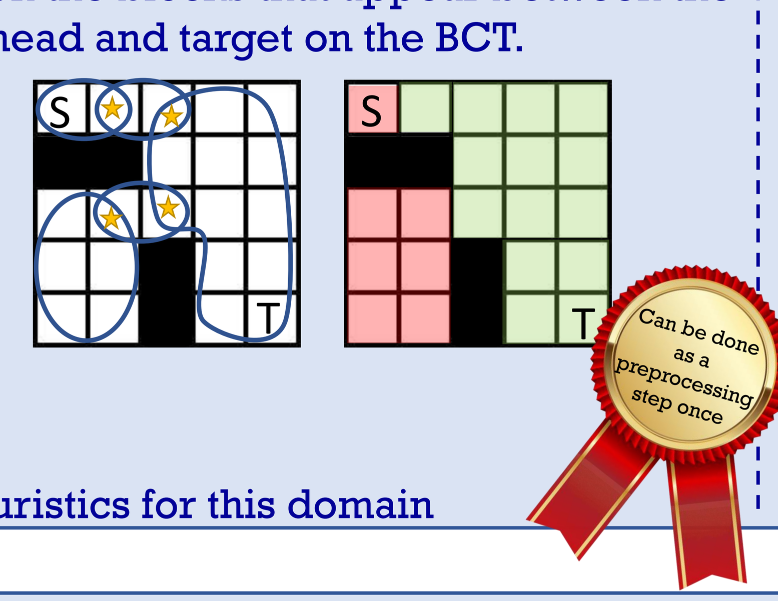
Reachable*

Count the remaining unvisited that we can reach.



Biconnected components*

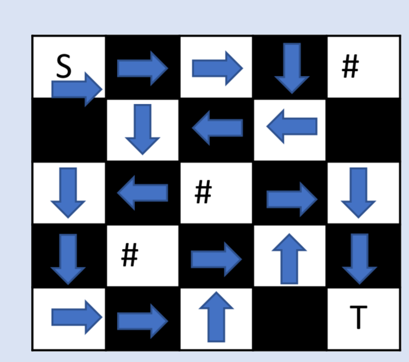
Count the remaining unvisited that are on the blocks that appear between the head and target on the BCT.



* - Known heuristics for this domain

Alternate Steps

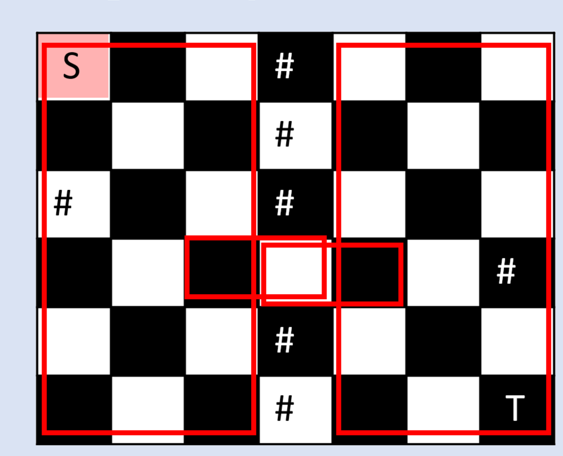
Improve heuristics on Bipartite graphs. Such as Grid and SIB.



Improve Reachable and BCC

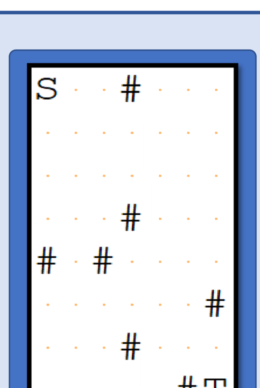
Separate Alternate Steps

Calculate the alternate steps on every BCC block separately.



Improve BCC even more

Results


Sample 7x8 grid

We used A* and DFBnB search algorithms and the above heuristics and pruning methods

Random Grids

Grid maps with random blocked cells - 360 maps with variety of blocked percentile

Heuristic	A*	A*+BSD	A*+RDP	BnB	BnB+BSD
R	45,211	21,815	16,494	49,772	24,823
R+ALT	34,271	15,708	14,051	38,100	18,286
BCC	8,366	2,703	2,271	9,623	3,447
BCC+ALT	7,491	2,187	2,077	8,651	2,869
BCC+S+ALT	7,348	2,097	**2,025**	8,499	2,771

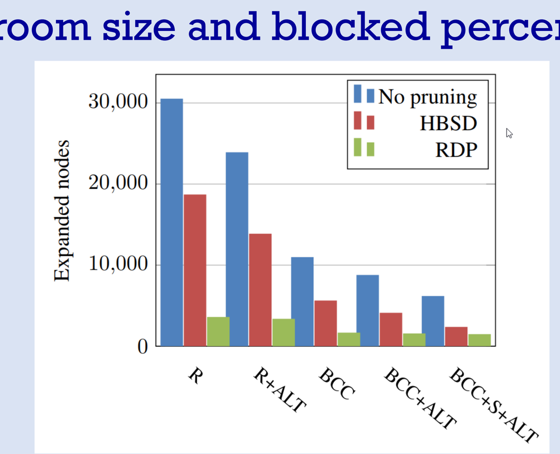
Expanded nodes

Heuristic	A*	A*+BSD	A*+RDP	BnB	BnB+BSD
R	10,808	2,626	17,576	601	448
R+ALT	6,155	1,417	14,386	463	338
BCC	377	110	294	195	105
BCC+ALT	300	**82**	262	150	86
BCC+S+ALT	311	86	261	149	94

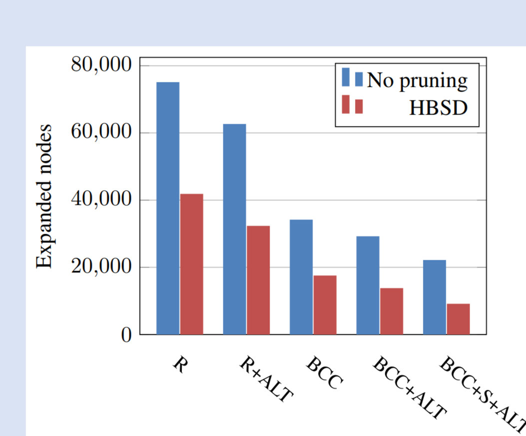
Runtime (ms)

Room Maps

Grid maps with rooms - 400 maps with variety of number of rooms, room size and blocked percentile

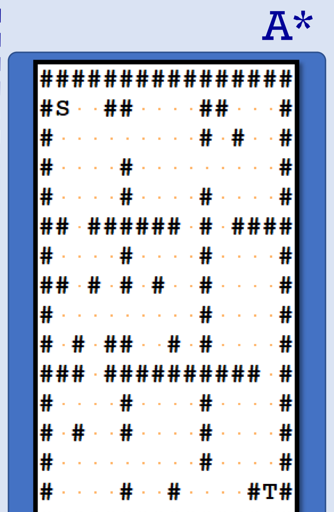


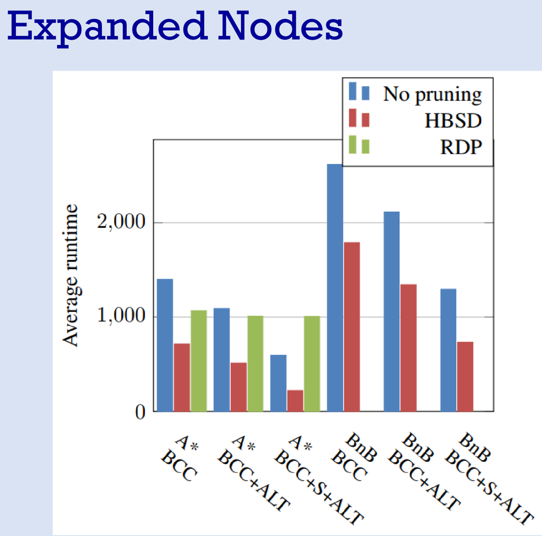
A*



DFBnB

Average Expanded Nodes


Sample room map with 3x3 rooms and 4x4 per room



Runtime (ms) – only for BCC based

Conclusions

- BCC Preprocessing *always* improve the runtime.
- Pruning scientifically improve the search time and nodes expanded
- Pruning effectiveness order:
  None ≤ BSD ≤ RDP
- Heuristics effectiveness order:
  R ≤ R+ALT ≤ BCC ≤ BCC+ALT ≤ BCC+Sep. ALT
- BSD has the fastest runtime and RDP has the strongest pruning ability - improvement of RDP runtime is the next thing for this domain.

GitHub

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