Divide and Conquer-Beam Stack Search

S Srinivas Saurab, CS16B039
Ganta Gowtham, CS16B008
Team-2

April 25, 2019

The Beam stack search:

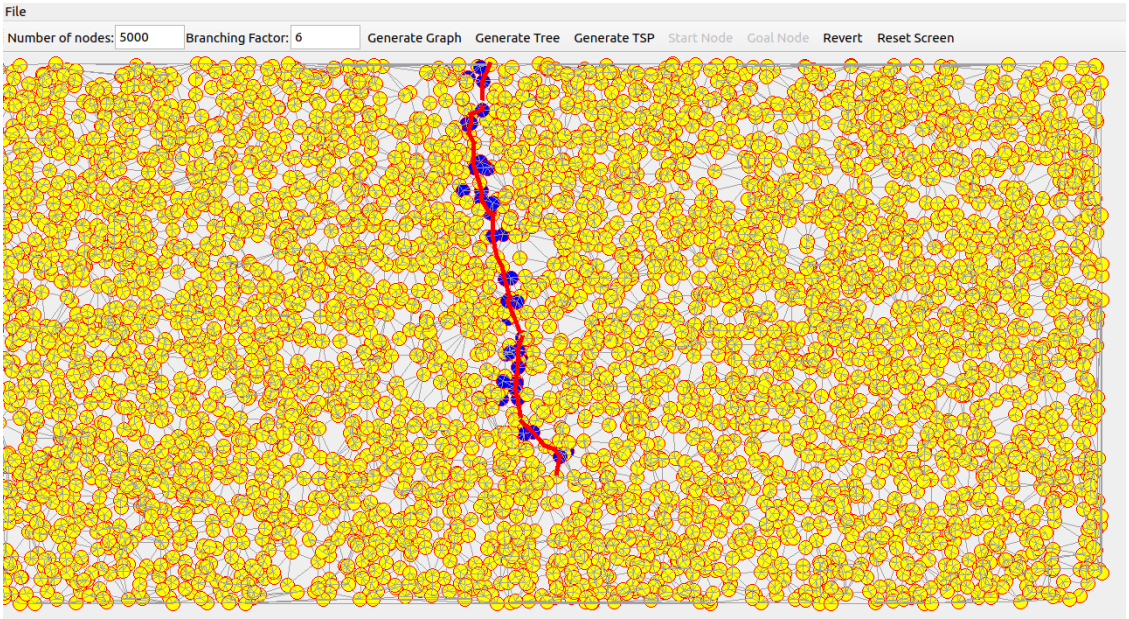
The beam stack search is an optimal path finding algorithm that is an advancement over Beam search. We know that Beam search over f-values is an incomplete algorithm. This is because it inadmissibly prunes the nodes that are costlier than the cheapest w-nodes in a layer (beam width : w). Beam stack search provides a workaround to this pruning by using an additional stack called BeamStack.

Backtracking and completeness:

The BeamStack allows the algorithm to regather the nodes lost during pruning. This is done in the process of backtracking. When the search algorithm reaches the last layer with no more successors, if it has not found the goal node yet, the algorithm backtracks upto the layer where at least one node valid has been pruned ($f(N) < U$). The algorithm now generates the next best w-nodes (or less) and starts to rebuild the tree from that layer again. This can be seen as DFS with some width to the search frontier. The BeamStack allows for regenerating the pruned nodes at all the levels starting from bottom to top.

The algorithm explores the graph starting from path which is a combination of least possible f-values. Later it keeps increasing the path-cost as it moves leftwards in the graph (if graph is sorted as per f-values). Due to this the algorithm explores all possible paths from [0,U] where U is the initial upper limit or the cost of the best path available so far. Hence the algorithm is complete.

/*Insert beam, beam stack diagrams*/



The divide and conquer-beam stack search:

The DC-Beam Stack Search is an extreme memory conserving version Beam Stack search. This algorithm stores only 4w nodes in its memory at any given time. These 4w nodes are :

1. Relay layer : These act as the fulcrum to regenerate solutions
2. The CLOSED layer (n-1)
3. The EXPANDING layer (n)
4. The OPEN layer (n+1)

The principle of finding the goal node is same as that in beam stack search. However, to reconstruct the path, only the relay layer can be used as there is no information about parent to each of the nodes.

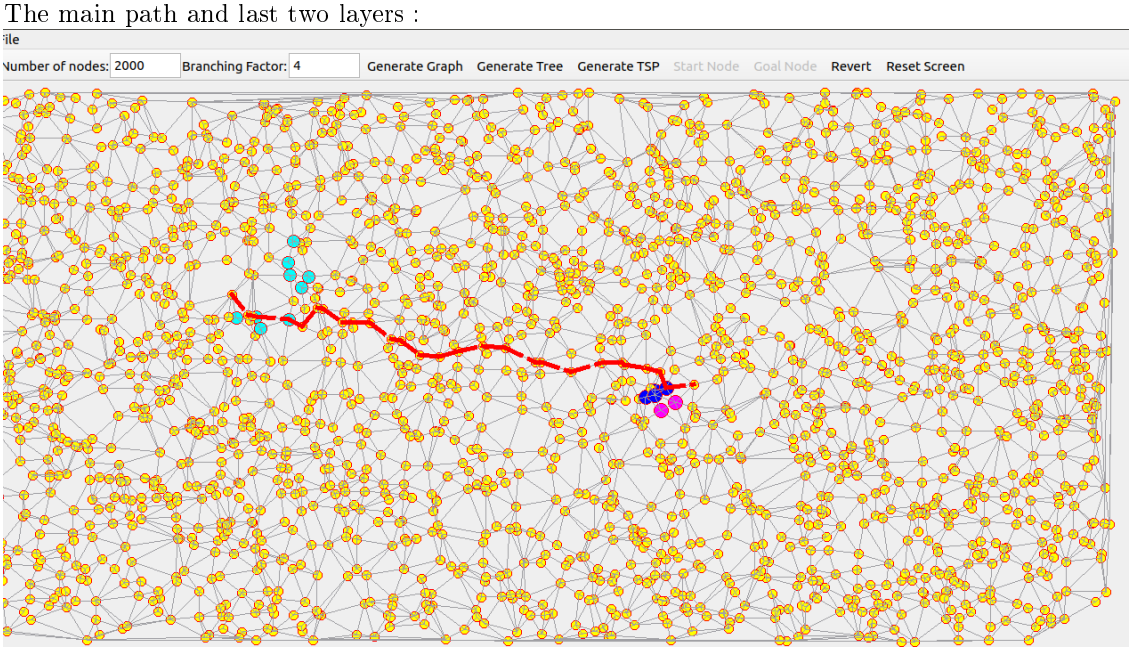
Path reconstruction:

Once we reach to a point beyond the relay layer, the association of a node with one of the nodes in the relay layer is propagated to it's children. This means that each node in the open layer and the expanding layer has a pointer to excatly one of the nodes in the relay layer (called relay ancestor).

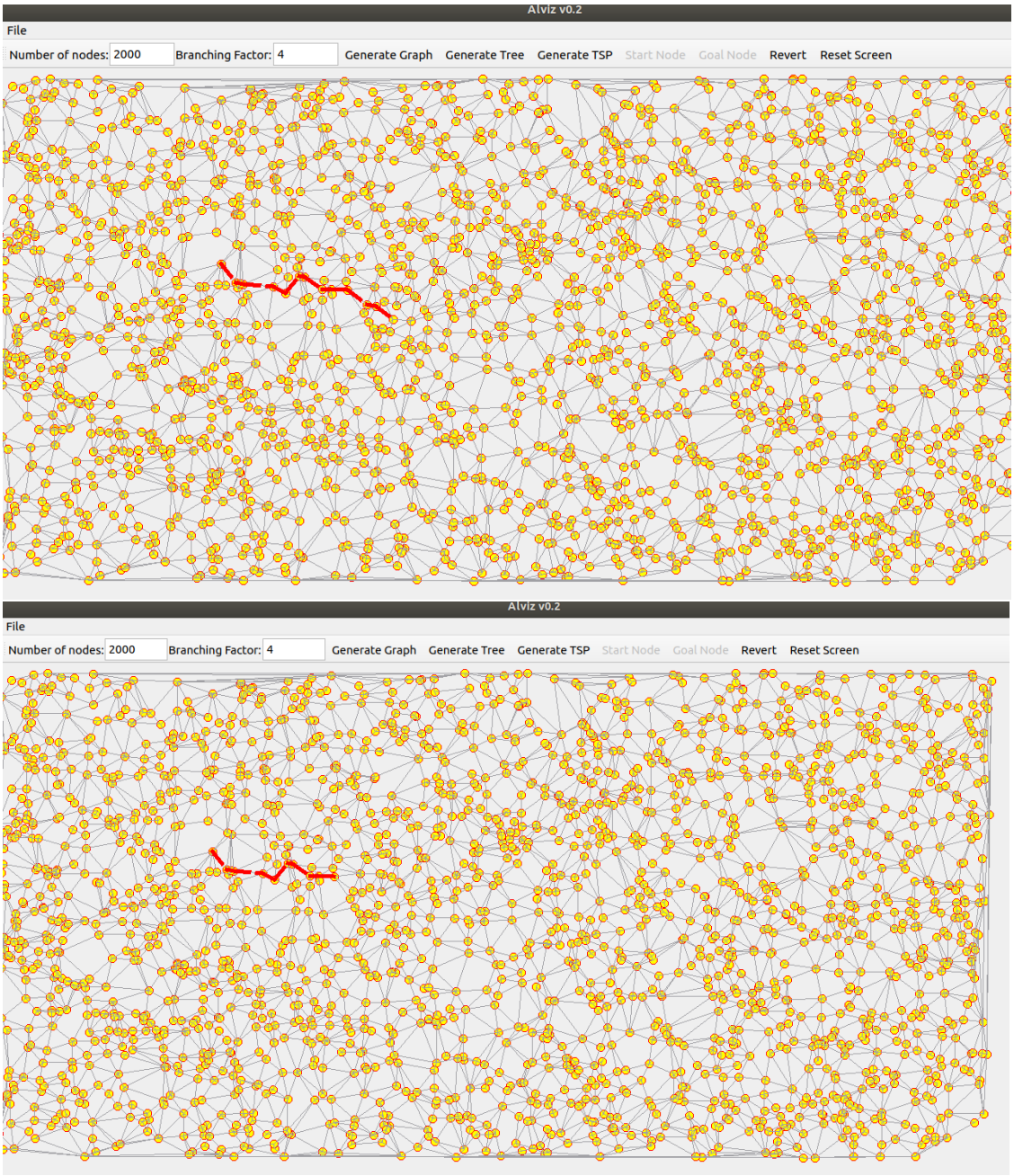
When the goal node is found, its relay ancestor is memorized. Now the algorithm is applied recursively, from Start to relay ancestor and relay ancestor to the goal node. This continues until the pairs become adjacent vertices in the

given graph. This corresponds to the base case of the recursion. An additional data structure is used to remember these base cases. Once the recursion terminates, these single length paths i.e edges can be used to reconstruct the given path from start node to the goal node.

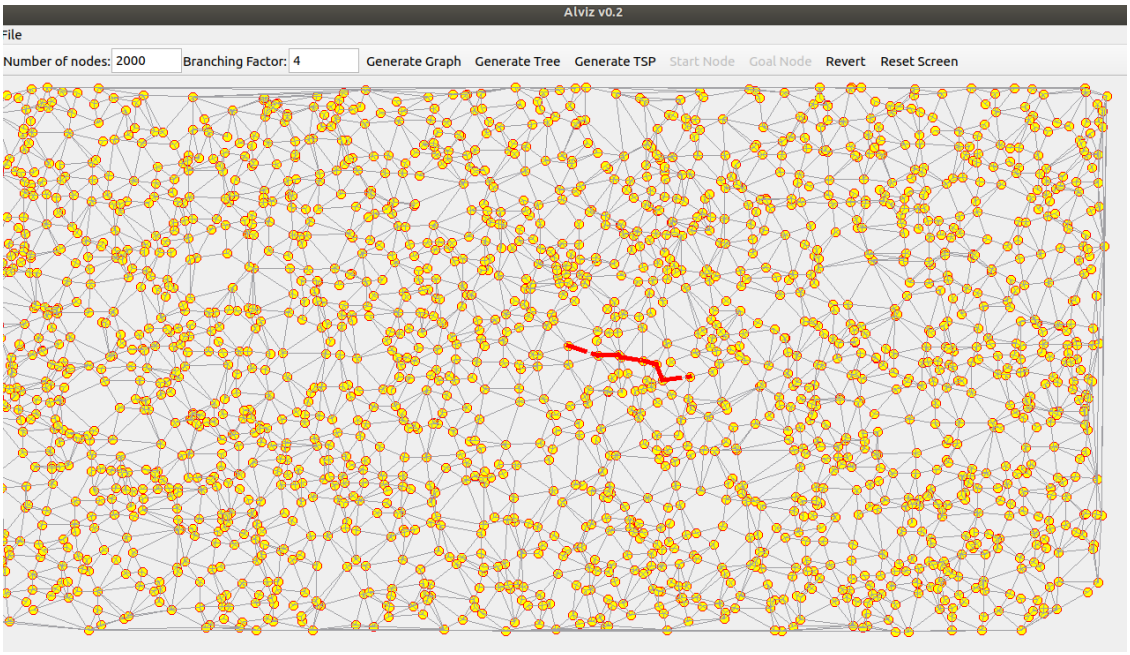
The main path and last two layers :



Showing the partial paths used in divide and conquer:


Alviz v0.2
File
Number of nodes: 2000
Branching Factor: 4
Generate Graph
Generate Tree
Generate TSP
Start Node
Goal Node
Revert
Reset Screen
Alviz v0.2
File
Number of nodes: 2000
Branching Factor: 4
Generate Graph
Generate Tree
Generate TSP
Start Node
Goal Node
Revert
Reset Screen


File
Number of nodes: 2000
Branching Factor: 4
Generate Graph
Generate Tree
Generate TSP
Start Node
Goal Node
Revert
Reset Screen
alviz10.py
Thu 23:51
Alviz v0.2
File
Number of nodes: 2000
Branching Factor: 4
Generate Graph
Generate Tree
Generate TSP
Start Node
Goal Node
Revert
Reset Screen


Alviz v0.2
File
Number of nodes: 2000
Branching Factor: 4
Generate Graph
Generate Tree
Generate TSP
Start Node
Goal Node
Revert
Reset Screen