Artificial Machine Intelligence Search Assignment

Jason Giancono (16065985)

May 11, 2014

Abstract

Different sets of data requires different search strategies. For the assignment, I implemented and tested three different search algorithms, Branch and Bound with Dynamic Programming, A* and Stochastic Hill Climbing. While Branch and Bound and A* both found optimal solutions for all the provided test cases, Stochastic Hill Climbing failed to find the global maximum on test cases which had more than one apex (local maximum).

Contents

1 Introduction

For my assignment I have implemented two Informed Search Strategies. The strategies used were Branch and Bound which is a search derived from Dijkstra's algorithm (Dijkstra, 1959) with Dynamic Programming (DP) (Bellman, 1954) and A* (Hart et al., 1968). I have also implemented the Stochastic Hill Climbing (Russell et al., 2003, 124) as a Local Search strategy.

Branch and Bound is a generic term for a class of algorithms, because the assignment did not specify which one to use, I have assumed it meant the Uniform-Cost Search (Russell et al., 2003, 83-85) which is described in the lecture slides as Branch and Bound, therefore I from now on will be referring to this as Branch and Bound.

2 Bugs/Limitations

2.1 Branch and Bound

Branch and Bound is functioning and will always find the optimal path if one exists. Due to the nature of Dynamic Programming where a path is discarded if the next node has already been reached with a smaller cost, no solution will ever have the same node revisited twice. This could be seen as a good thing as you usually wouldn't want to backtrack, but it is also a limitation as some alternative paths which don't backtrack will not be discovered.

The algorithm will come up with alternate paths, but only ones where the paths diverge and do not merge until they get to the goal node (because if they merged, the path would be thrown out).

Both Branch and Bound and A* use the same data structure, which is basically a Linked List of nodes which contain the name, the lowest cost to node so far (only used in the Dynamic Programming B&B), the heuristic and a Linked List of nodes they are connected to, where each node in that list also contains the weight of the link. One limitation on this structure is that it assumes the graph is static and will not change. Changing the graph would be problematic and you would need to find every connected node pointer if a node was removed.

2.2 A*

The A* search is working. Unlike Branch and Bound, I did not think it was necessary to include a dynamic programming feature like in Branch and Bound, which means that the A* search will eventually re-visit nodes. If the heuristics are consistent, then it should find the optimal path before nodes are re-explored. However, when finding alternate paths, nodes are sometimes re-explored, especially if they have a small link cost coupled with a larger heuristic difference. This also means the algorithm will keep coming up with alternate paths if you

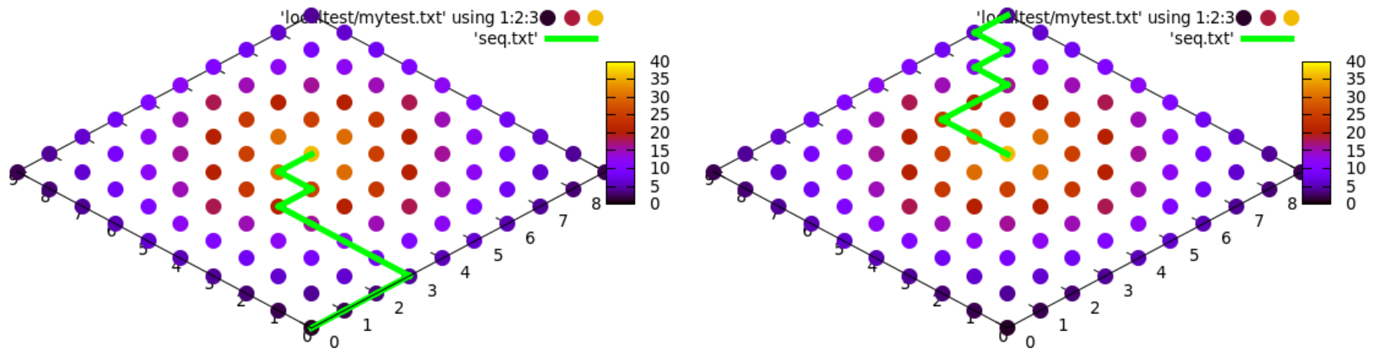

Figure 1: The Stochastic Hill Climbing works well on single apex maps.

ask it to, with the resulting paths just being previous ones but with nodes revisited.

Some of the supplied test cases had heuristics which weren't consistent which meant that nodes were revisited before the goal was found.

2.3 Stochastic Hill Climbing

Stochastic Hill Climbing is not a complete search, and because I did not want to implement what could be considered a different algorithm, I did not add Simulated Annealing (Kirkpatrick et al., 1983) to the algorithm which would have improved it. The incompleteness causes the algorithm to perform poorly when given maps with multiple local maximums.

The algorithm originally returned the current node as maximum if it reached a plateau, however I added in an option to prompt for a certain amount of consecutive sideways moves before the search begins. This allows the user to select an appropriate amount for their map.

The algorithm by default starts in the corner with the lowest x/y value, it can also be provided with start coordinates if the user wishes. I did not implement random restart (Russell et al., 2003, 124) into the algorithm because I thought it did not suite the problem. Due to the way I implemented the data structure of the map, you would have had to navigate through it to get to a random spot, defeating the purpose of the search. I chose to store the data in a linked-list style structure because it made it easier to sort using insertion sort, than if I were to do it in arrays.

3 Local Search Performance

3.1 Simple Maps (Single Apex)

The algorithm performs perfectly on single apex maps, reaching the global maximum every time as seen in Figure 1. Due to random nature of the algorithm it does not find the most optimal path for these maps. A steepest-ascent search (Russell et al., 2003, 122) can be more efficient for achieving optimality on single apex maps but it fails more often on more complex maps. Another thing which could have improved the optimality of the solutions for simple maps, is to have increased the chance of selecting a node based on it's steepness. This was not implemented in the algorithm due to the negative effect it tends to have on multiple apex maps. Instead the algorithm uses the first-choice method (Russell et al., 2003, 124) of stochastic hill climbing

3.2 Complex Maps (Multiple Apex)

When testing on complex maps with multiple apexes, the algorithm will seldom find the global maximum. This could have been improved by implementing a random restart (Russell et al., 2003, 124) in the hill climbing but this did not suite the application as you are provided discrete coordinates that cannot be randomly generated. The algorithm does sometimes find a solution close to the global maximum which could be sufficient depending on the application.

Figure 2 shows several different maps where the algorithm was tested with several different starting points. As you can see, without Simulated Annealing, the algorithm tends not to make it to the global maximum. Over a test of thousands of tries, assuming there is a path to the maximum that does not consist of downhill steps, the global maximum would be found, but this is not an efficient way of searching.

4 Conclusion

Branch and Bound with DP is a good search algorithm for finding the optimal solution if we don't have a good heuristic measure. A* on the other hand will beat Branch and Bound if there is a heuristic that is admissible and consistent it will do better than Branch and bound. If the heuristic is not admissible and consistent Branch and Bound with DP will outperform it.

Stochastic Local Search is suited for searches where there are many (or unlimited) neighbouring nodes (Russell et al., 2003) which means that the algorithm will rarely find itself in a position where there are no uphill nodes until it is very close to the global maximum. Navigating a 3D map does not suite the stochastic local search due to the fact that 3D maps generally have many local maxima. Because of the way the algorithm runs by reading in an entire file the whole search is O(n) anyway which defeats the purpose of doing the search in

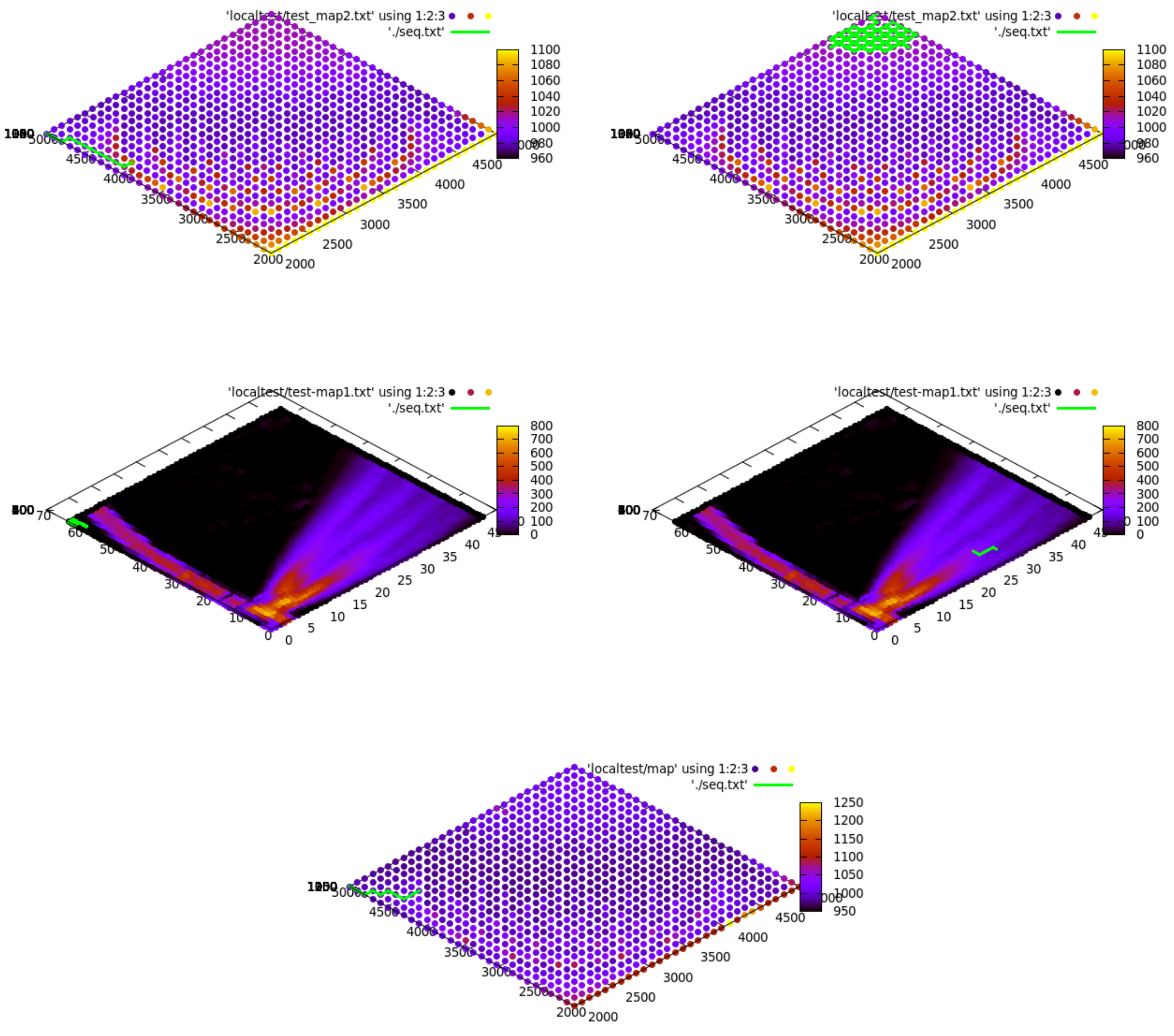

Figure 2: The Stochastic Hill Climbing rarely finds the global maximum on maps with multiple local maximums.

the first place. A better way to structure the data would be in a mark-up file in order to not require the entire file to be loaded into the program

References

Bellman, R. (1954, 11). The theory of dynamic programming. *Bulletin of the American Mathematical Society 60*(6), 503–515.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271. 10.1007/BF01386390.

Hart, P., N. Nilsson, and B. Raphael (1968, july). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on 4*(2), 100 –107.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). Optimization by Simulated Annealing. *Science, Number 4598, 13 May 1983 220, 4598*, 671–680.

Russell, S. J., P. Norvig, J. F. Candy, J. M. Malik, and D. D. Edwards (2003). *Artificial Intelligence: A Modern Approach 3rd Ed.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc.