CS 830 Intro to AI, Spring 2025

Written Assignments, Sushma Akoju

Assignment 2

https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
https://www.redblobgames.com/pathfinding/a-star/introduction.html
https://github.com/aimacode/aima-python

1. Describe any implementation choices you made that you felt were important. Clearly explain any aspects of your program that aren't working. Mention anything else that we should know when evaluating your work.
   I created a node, graph, problem classes to record and separate the node & states, problem space, dirt cells, charge, and recharge stations.
   The heuristics I chose were manhattan distance, max(dx,dy) and min(manhattan distance, max(dx,dy)) just to see the effects of dominating heuristic. I created a Node class to hold all the node related information. I created a Problem class, to store all problem related global information including the final results returned by the search algorithm.

   The battery option heuristic does not work in my code, it runs longer than 30 mins for a 5*5 grid with 3 charge stations and 1 blocked cell and 1 dirt cell.

   I generated world data with max blocked cells. Just like small1.vw world provided to us, there were small number of valid paths to the dirt cells when the number of blocked cells are greater than 20% of R*C grid cells. DFS does not seem to ever find the goal state as only half the dirt cells get cleaned on this data with >20% blocked cells/walls. The same case with UCS and GBFS. Only A* and IDA-STAR can navigate and find optimal paths or stop recursions. IDA -STAR seems better at the optimal paths due to the threshold but it seems IDA expands the same state multiple times but with

different costs. Most of the implementations or pseudo code assumes the graph adjacency matrix is represented as a Graph data structure. But I did not use a Graph data structure.

About processing command line arguments in python:
I used the python Argparse library (created by one of NLP professors). However, with sys.argv or Argparse, the arguments are processed differently by Python. For example “vw-validator -time 5 -- ./run.sh a-star h1 < tiny-1.vw” will not work. Python sees 9 arguments which are not correct. To run my script, I had to use
“vw-validator -time 5 -- ./run.sh a-star h1 ./worlds/tiny-1.vw” and it worked for me.
My code would run for and for all small world simulations:

./run.sh a-star h1 ./worlds/tiny-1.vw
./run.sh ida-star h1 ./worlds/tiny-1.vw
./run.sh dfs h1 ./worlds/tiny-1.vw
./run.sh uniform-cost h0 ./worlds/tiny-1.vw
./run.sh greedy h0 ./worlds/tiny-1.vw
./run.sh bfs h0 ./worlds/tiny-1.vw
./run.sh dfs h1 ./worlds/tiny-1.vw

I was able to run my script on all world data except hard1 and hard2 world simulations as they ran longer both on gpu (google colab) or cpu.

I tried to go to professors and TA office hours, but both are intersecting with my classes/my TA office hours.

2. Explain each heuristic function you devised and prove that each is admissible.
   The first heuristic Manhattan distance (closest dirt cell from this cell).
   The second heuristic using max of closest dx or dy.
   The third heuristic used was min(manhattan, max(dx,dy)).

   Proof for Admissibility of Manhattan distance as heuristic:
   D(node, goal) = | node.x - goal.x | + | node.y - goal.y |
   Clearly, so far there are no obstacles, it seems like a straight line distance, in terms of coordinates in the grid. If there are obstacles, then the optimal

path would increase the path length and hence it cannot be worse than the original straight line distance. Manhattan distance can never overestimate the true cost to the nearest dirt cell. therefore it is admissible.

Verifying consistency:
For a given node n, cost(n, n-1) is the cost between n to n-1, where n-1 is the parent.
If h (n-1) - h(n) <= cost(n, n-1) => h (n-1) <= cost(n, n-1) + h(n)
Since h(n) is admissible and is <= path from n to goal, h(n-1) <= cost(n, goal).
This proves that Manhattan distance is admissible.

By IH:
h(goal - k) <= h*(goal-k), where k is some arbitrary node.
Inductive step:
h(goal - k) <= cost(goal-k-1) + h(goal-k) <= cost(goal-k-1) + h*(goal-k)
h(goal - k) <= cost(goal-k-1) + h*(goal-k)
And cost(goal-k-1) + h*(goal-k) = h*(goal-k-1)
h(goal - k) <= h*(goal-k-1)
By IH, this holds for all nodes, proves consistency which implies admissibility holds.

Proof for Admissibility of max dx, dy distance as heuristic:
dx = | node.x - goal.x |
dy = | node.y - goal.y |

D(node, goal) = max { dx, dy }
In comparison to Manhattan distance, D(node, goal) is always less than Manhattan distance. Therefore, since Manhattan distance is already consistent over any arbitrary node and hence is admissible, max(dx,dy) cannot overestimate the closest distance to the nearest dirt cell either in x direction or y direction.

Proof for Admissibility of min (Manhattan, max(dx, dy)) distance as heuristic:

Although this is implicit, manhattan distance is always going to be greater than max(dx,dy) since manhattan distance = dx+dy, the min function would always seem to yield max(dx,dy). I mainly created to test the dominating

heuristic concept (example: in “guessing” the relevance of a document to the text-based query in Natural Language Processing).

The min function described here follows the dominating heuristic where for all n, h1(n) >= h2(n) i.e. Let h1 be Manhattan distance heuristic and h2 be max(dx,dy) heuristic.

h1(n) >= h2(n) and there exists n such that h1(n) > h2(n). Therefore h1(n) is a better choice.
And considering h3 = min(h1, h2) which results in h2 always, for all h1(n) >= h3(n) and there exists a node where h1(n) > h3(n). Therefore since h1 is dominating, by theorem, A* cannot expand nodes greater than both h2 and h3 respectively.

3. What is the time and space complexity of each algorithm you implemented? Which algorithms are admissible?

   I implemented DFS, BFS, UCS, A-star, GBFS, IDA, IDA-STAR.
   DFS is not admissible. But all others seem fine. BFS is admissible only if ops cost is 1 (referred from lecture notes).

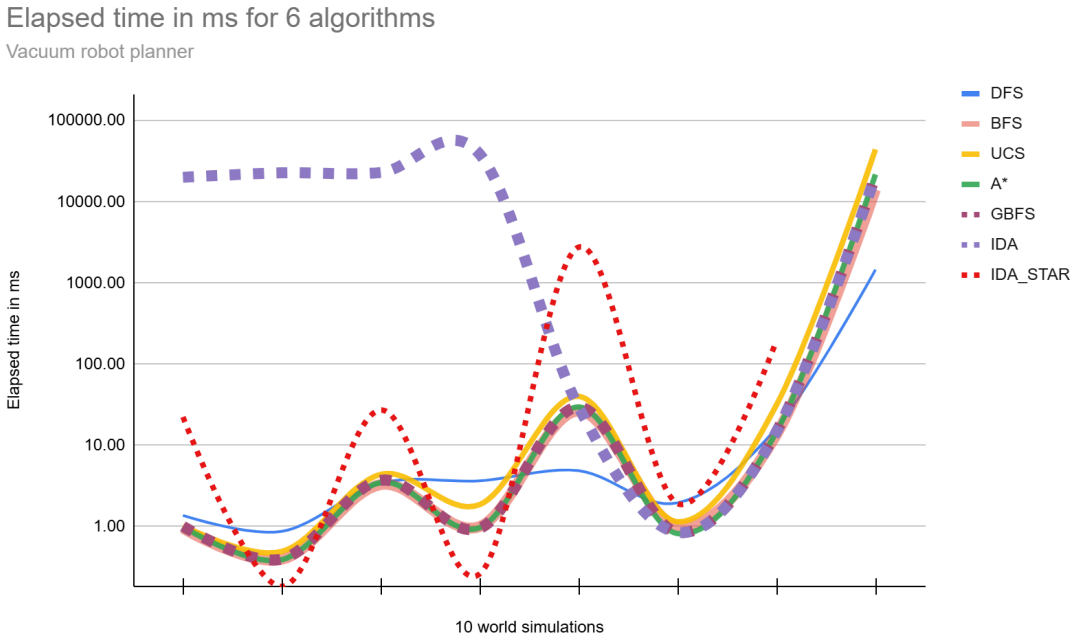
   The time complexities:
   Let d be the solution depth, b branching factor and m max depth (edge branching factor). I referred following research publications:
   1. https://www.sciencedirect.com/science/article/pii/S0004370201000947?via%3Dihub (I was searching for proofs for optimality o A* and IDA-star, hope they seem relevant)
   2. https://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/astar.pdf (I was searching for proofs for optimality o A* and IDA-star, hope they are relevant)
   3. https://www.sciencedirect.com/science/article/pii/0004370285900840?via%3Dihub (piazza reference from steve)
   4.

Algorithm	Time Complexity	Space complexity
DFS	O(b^m)	O(bm)

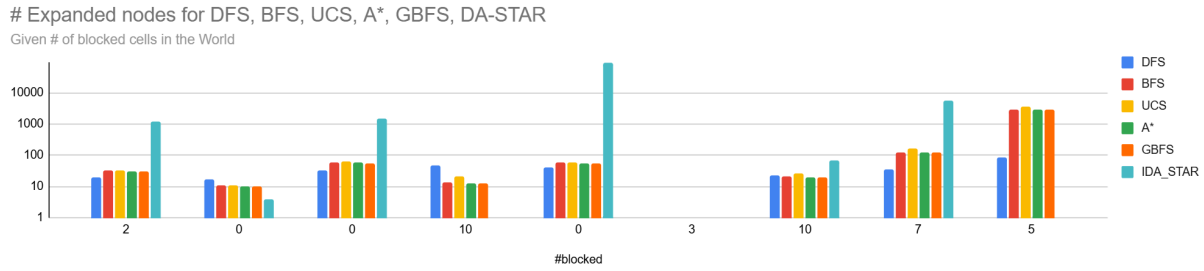
BFS	O(b^d) All nodes up to d or b+b^2+b^3...+b^d	O(b^d)
UCS	O(b^d)	O(b^d)
A*	O(b^d)	O(b^d)
GBFS	O(b^m)	O(bm)
IDA-star	~ O(number of node expansions)*	O(|N|) N, where N is the number of nodes in the graph after cutoff O(d)
DFID	O(b^d (1-1/b)^-2)	O(d)

4. Provide empirical results confirming your answers to the previous question.

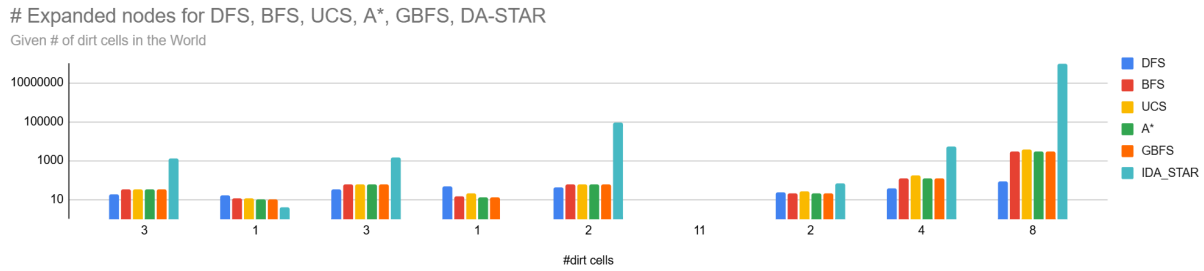
The elapsed time in ms for 6 algorithms over 10 world simulations



The number of expanded nodes given number of blocked cells



The number of expanded nodes given number of dirt cells



5. What suggestions do you have for improving this assignment in the future?

The time and space constraints exist for running larger world simulations. The hard1 and hard2 simulations ran for more than 2 hours on gpu. Maybe an LRUCache or transposition table with tabulation hashing/zobrist hashing like the one used for board games or chess or go states. (table lookup with exclusive OR from universal family of hash functions such as an irreducible polynomial for better collision resistance, also often used for hashing in maintaining cloud/other recent online file systems)