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.

Aside from the vw-validator not working with memory utilization due to the JVM, I feel like my design of h2 heuristic could be explained in detail. The overall premise of it is to represent the minimum spanning tree of dirt locations in a relaxed state of the world, where the robot starts on a dirt, can go through any blocked cells if present. Since the MST, using Manhattan distance from dirt to dirt as the cost would be the optimal method to visit all nodes sequentially in any graph, and this relaxed graph, I figured it would be a highly selective heuristic.

Given implementing a MST would've entailed a lot more trivial code ie disjoint sets, prims algo, etc. and the costs would be computationally expensive, I implemented it in a more simplified manner. It simply computes the Manhattan distance from each node to the next, assuming the robot starts on a node, and if a dirt is within said Manhattan path it adds 4 to the overall h value as it takes 4 actions to go around a dirt, assuming no dirt are next to said dirt. With all this said, I would title h2, the MSTish heuristic, I hope that gives some insight to my thought process.

2. Heuristic Admissibility proof:

h1:

Relaxed Problem: Robot can teleport, no need to Vacuum

h1: Counts # of Dirt in State

Proof by Induction: $h(n) \leq h^*(n)$

Base case: P' (Relaxed problem)

Let n=0, n is Dirt in state

$h(0)=0$ thus $h(0) \leq h^*(0)$
$h^*(0)=0$

Induction Step:

Assume $h(k) \leq h^*(k)$ for n=k

$h(k+1) \leq h^*(k+1)$
$h(k+1)=(k+1)$ thus $h(n) \leq h^*(n)$
$h^*(k+1)=k+1$

1b.) Opt. Solution for Relaxed Problem $P' \leq$ Opt. sol for actual Problem P

i = one dirt

$P'(i)=0$ $\quad P'(i) \leq P(i)$
$P(i) \geq 1$ $\quad$ for $i=0$

Inductive step

$P'(k+1)=k+1$ $\quad$ so $P'(n) \leq P(n)$
$P(k+1) \geq \underset{\text{Vacuum action}}{k+1+1}$ $\quad$ thus $h(n)$ of $P' \leq$ $h^*(n)$ of P

H2

$$H2: \sum_{n=0}^{n=\#\text{ of Dirt}} (\text{Manhattan Distance}(\text{Point}_n, \text{Point}_{n+1})$$

Base Case: n=2

$h(0) = Abs(P_1.x - P_2.x) + Abs(P_1.y - P_2.y)$

$h^*(0) = \quad \downarrow + \underset{\text{distance to Dirt}}{Z}$ $\quad h(n) \leq h^*(n)$ for $n=0,1,2$

Inductive step

$$h(k+1) = \sum_{n=0}^{n=k+1} Abs(P_n.x - P_{n+1}.x) + Abs(P_n.y - P_{n+1}.y)$$

$h^*(k+1) = \quad \downarrow + \underset{\text{distance to Dirt}}{Z}$ $\quad$ so $h(n) \leq h^*(n)$

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

A-Star

Admissible: Yes. If the heuristic is admissible, and truly not overestimating then at the end of the day A-star will consider all nodes exactly as UCS does and will return the optimal path.

Time Complexity: The worst-case time complexity will assume the heuristic is 0, then the time complexity will perform exactly like UCS or O(b^d). Yet case by case the heuristic will change this. The time complexity of the heuristic computation should also be factored in.

Space Complexity: As A* stores all generated nodes regardless of h-value it will have the same space complexity as UCS or O(b^d).

Iterative Deepening Depth First Search:

Admissible: Yes. As IDDFS simply runs DFS incrementally and DFS will search the whole tree to the specified depth the first goal state it will find will be the shallowest.

Time Complexity: Given that there is a solution the ID-DFS will search, but not store all nodes in the search tree to the given depth d. This results in a time complexity of O(b^d)

Space complexity: As it simply preforms DFS incrementally, and each iteration the current stack is overwritten to accommodate all the nodes at the next increment depth it will use O(bd) complexity where d is the goal state depth instead of o(bm) where m is the max depth like DFS does.

Previously proven correct info (by proven correct I am assuming its correct given I received no point deductions, and my own confidence in my work)

Uniform Cost Search:

Admissible: Yes. UCS always returns the lowest cost path to a goal state

Time Complexity: As UCS will explore every node with a lower g-value up to a max g-value of say d which the goal state lies, and each node can have b succesor states, in this case b being 5 the time complexity will be O(b^d) or O(5^d).

Space complexity: Since the entire state space up to the max g-value of d will be stored in the closed list, holding all other storage requirements constant(ie state representation, node representation, etc.), given b succesor states the space complexity will be O(b^d) or O(5^d) in this case.

Depth First Search:

Admissible: No. Depth First Search while with cycle checking will return a path, will simply return the first path found, not guaranteed to be the shortest.

Time complexity: DFS will explore in the worst case all nodes in the state-transition-graph through the entire depth of the graph. Given a branching factor of b and depth of m DFS would have a time complexity of O(b^m) or O(5^m).

Space Complexity: Given a maximum path length of m each nodes' children must be stored for future access in a stack. Given a branching factor of b this results in a space complexity of O(bm).

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

Regarding admissibility and efficiencies of A* h1 It is clear it generates less nodes and works in a faster amount of time than UCS

File, A*: nodes generated, h(1), UCS: nodes generated

Tiny-1 A*: 25 UCS:25

Tiny-2 A*: 139 UCS: 164

Small-1 A*: 1236 UCS: 1912

Hard-1 A*: 4129 UCS: 5058

Hard-2 A*: 103731 UCS: 566058

Regarding admissibility and efficiencies of A* h2 It is clear it generates less nodes and works in a faster amount of time than UCS with the same path length

File, A*: nodes generated, h(2), UCS: nodes generated

Tiny-1 A*: 25 UCS:25,

Tiny-2 A*: 96 UCS: 164,

Small-1 A*: 632 UCS: 1912

Hard-1 A*: 1531 UCS: 5058 Path

Hard-2 A*: 33588 UCS: 566058

Regarding admissibility it is clear UCS provides the least cost solutions while DFS simply provides a solution:

File UCS: path length DFS: path length

Tiny-1 UCS:4 DFS: 6

Tiny-2 UCS:9 DFS:23

Small-1 UCS:24 DFS:119

Hard-1 UCS:59 DFS:465

Hard-2 UCS: 141 DFS:3674

Regarding Time complexity DFS begins to outperform UCS in larger state spaces. While depth d of UCS is always less than max depth m of DFS, at greater depths m DFS will be faster due to its non-admissibility and non-requirement of searching by surrounding nodes.

Tiny-1 UCS:0.693 seconds DFS:0.747 seconds

Tiny-2 UCS:0.747 seconds DFS: 0.738 seconds

Small-1 UCS:0.778 seconds DFS: 0.757 seconds

Hard-1 UCS: 0.790 DFS: 0.822 seconds

Hard-2 UCS: 13.880 seconds DFS:1.259 seconds

Unfortunately, I was not able to get the validator to output the correct memory utilization for my solution. Despite this fact within my implementation in DFS as only visited nodes and their direct successors, not the complete search space above a goal state depth its space complexity should be less than the UCS. This can further be proven by the node generated counts. As the node generated counts is directly proportional to states stored by both algorithms it is clearly shown DFS generates less nodes at the very least pointing towards less space used:

File UCS: nodes generated DFS: nodes generated

Tiny-1 UCS: 25 DFS:14

Tiny-2 UCS: 164 DFS:47

Small-1 UCS:1912 DFS:386

Hard-1 UCS:5058 DFS:1723

Hard-2 UCS:56,609 DFS:12,482

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

Possibly a guide to how to get correct validator output for space and time in recitation. I was not able to figure out, and it seems like others' as well to get the validator to output the correct space utilization numbers.