Rush Hour State Space Search Analysis

SOEN 472: Artificial Intelligence

Mini-Project 2

By Jonathan Hubermann 40125401 & Oliver Hassan 40132324

1. Length of the Solutions across Algorithms & Heuristics

	Average	h_1	h_2	h_3	h_4
UCS	3.2195				
GBFS	4.878	5.0488	5.0488	5.0488	4.3659
A/A*	3.2317	3.2195	3.2195	3.2683	3.2195
ALL	3.9621	4.1341	4.1341	4.1585	3.7927

- UCS had the shortest average length of solutions (follows from admissibility)
- Closely Folled by A/A*
- GBFS had the longest average length of solutions which follows as it tries to optimize the execution time
- Across Algorithms, the length of solutions between heuristics was closely related

- A VERY important note to make is that our chosen h4 heuristic provided the lowest average length of solution path among all heuristics (and this extended through the GBFS and A/A* algorithms)

1. Lowest-cost solutions

While travelling the data, lowest cost solutions were found using:

- UCS
- A* with h_1, $\mathbf{h_2}$ and h_4
- Puzzle 16 and 30 in data file showed differences between lowest cost solution with UCS (cost of 5) and A with h3 (cost of 6). Hence why h_3 is not included

1. Length of the Search across Algorithms & Heuristics

	Average	h_1	h_2	h_3	h_4
UCS	227.4				
GBFS	62.365				
A/A*	88.745				
ALL	92.4267	77.41	77.38	60.37	86.7

2. Admissibility & Optimalism Influence of each Heuristic

H_1 Heuristic

H_1 = number of vehicles blocking the exit

- **Pros:**
  - Well Informed heuristic
- **Cons:**
  - Does not consider blocked cars in front of the ambulance
- **Admissibility ✅**
  - The number of vehicles blocking the exit is always smaller or equal to the minimum solution path length
  - Thus, $h_1(n) \leq h_1^*(n)\ \forall n$

2. Admissibility & Optimalism Influence of each Heuristic

H_2 = number positions blocking to the exit

- <u>Cons:</u>
  - Less informed than h1 (more backtracking)
- <u>Admissibility</u> ✗
  - Consider the case where there is a horizontally oriented vehicle between the ambulance and the exit
  - Hence, $h_2(n) \not< h_2^*(n)$ $\forall n$
- Similar to H_1 (horizontal case diff)

H_2 Heuristic

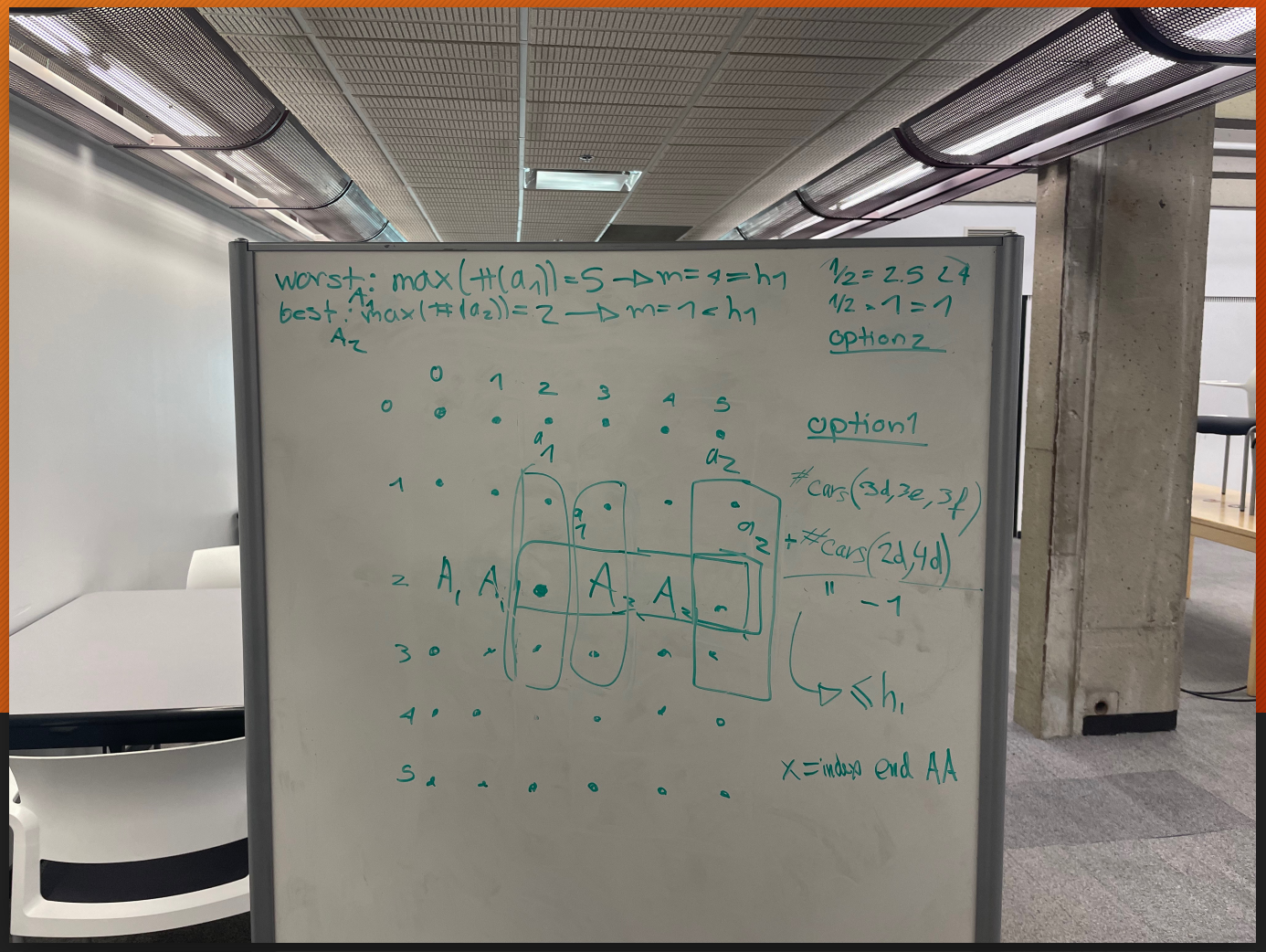
2. Admissibility & Optimalism Influence of each Heuristic

H3 Heuristic

H_3 = value of h_1 * λ , *where λ = 3*

- It sorts values just like H_1!

- Admissibility ✗
  - Consider the case where there is only one car blocking the ambulance and this car is able to move out the way



Brainstorming for H_4

Coming up with H4

	a	*b*	*c*	*d*	*e*	*f*
1	I	I	B	.	.	.
2	C	.	B	H	H	H
3	C	.	A	A	D	.
4	.	.	.	.	D	.
5	E	E	G	G	G	F
6	.	.	.	.	.	F

H_4 = (number of vehicles blocking the exit + number of vehicle on the bottom right + number of vehicle on the top right corner of the ambulance) * λ

where λ = 1/2

2. Admissibility & Optimalism Influence of each Heuristic

- **Pros:**
  - Takes into account more information than h1

- **Admissibility ✅**
  - Because h_1 is admissible
    and $h_4(n) \leq h_1(n) \leq h_1^*(n)\ \forall n$

Detailed example showing this can be provided during demo

H_4 Heuristic

3. Execution Time across Algorithms & Heuristics

	Average	h_1	h_2	h_3	h_4
UCS	16.778				
GBFS	1.7271	1.769	1.7474	1.7474	1.6446
A/A*	2.8591	2.837	2.8932	1.5988	4.1074
ALL	3.9025	2.303	2.3202	1.6731	2.876

Execution time fastest -> GBFS (Closely followed by A/A*)
Execution time slowest -> UCS

Using h_3 resulted in the fastest executions

Different search algorithms involve tradeoffs between execution speed and solution's optimality. As shown in the previous table informed searches were executed on average in a fraction of the time of uninformed searches (UCS).
However, it is not necessarily the case that an informed search is faster. It takes time to calculate the heuristic estimates, thus, if this calculation is easy to compute, an informed search algorithm will be faster than the uninformed.

Is an Informed search necessarily faster?

Other Functionalities and Analysis

- This assignment presented a very unique opportunity to challenge ourselves to use a data structure to represent the RushHour that enables us to apply any operations, indexing, and iterations in the most efficient manner in order to reduce execution time and reduce LINES OF CODE!!!
- As a result, a large portion of this assignment work was devoted to understanding and learning Numpy in order to store the entire game within a matrix that enables fast and efficient slices and operations executed in C language.
- For example, our entire code was made in a modular, and thus we avoided repeating redundant code by, say, slicing columns into rows and applying the same operations (regardless of horizontal or vertical orientation) while maintaining references to the original matrix in order to only write the algorithm for finding children and applying these displacements once.

Any Questions?