Lab 3: (Un)Informed Search Algorithms

Artificial Intelligence

Eduardo Vaca
A01207563

In this lab we had to implement three different search algorithms:

1. Uninformed Cost Search
2. A* with consistent heuristic
3. A* with inconsistent heuristic

For the A* problems we had to come up with our own heuristics. The ones I used were:

Consistent Heuristic

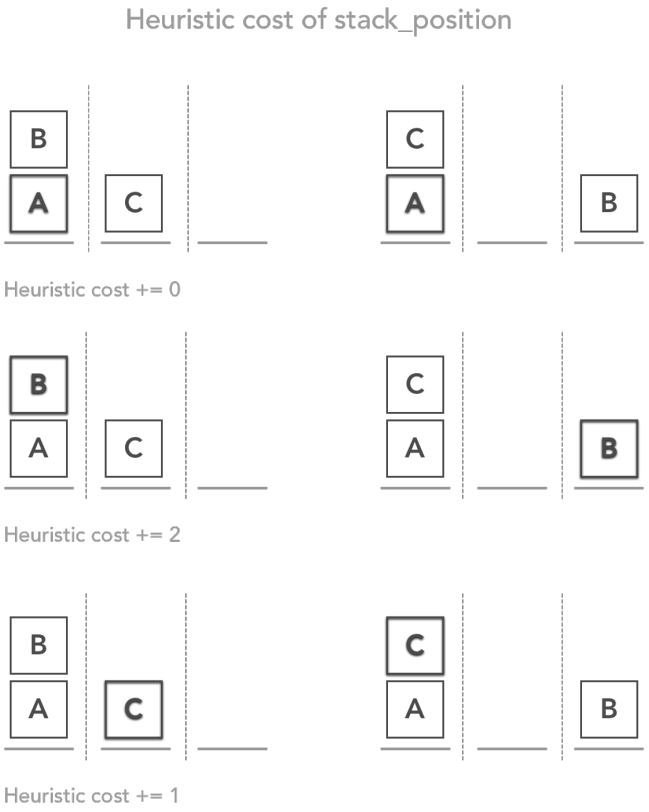
For the consistent heuristic I decided to analyze the actual position of the elements and compare it with the goal state elements. In other words, I obtain the heuristic cost of a current state by comparing:

- The stack index of all the elements with the elements of the goal state
- The height index of all the elements with the elements of the goal state

For example if in the current state, element A is on stack index 0 and in the goal state element A is in stack index 3, then we add to the heuristic cost: *abs(0 – 3)*. And if A has a height index of 1 and in the goal state it has index 0 then we add to the heuristic cost: *abs(1 – 0)*.
For better understanding here is a visual example:

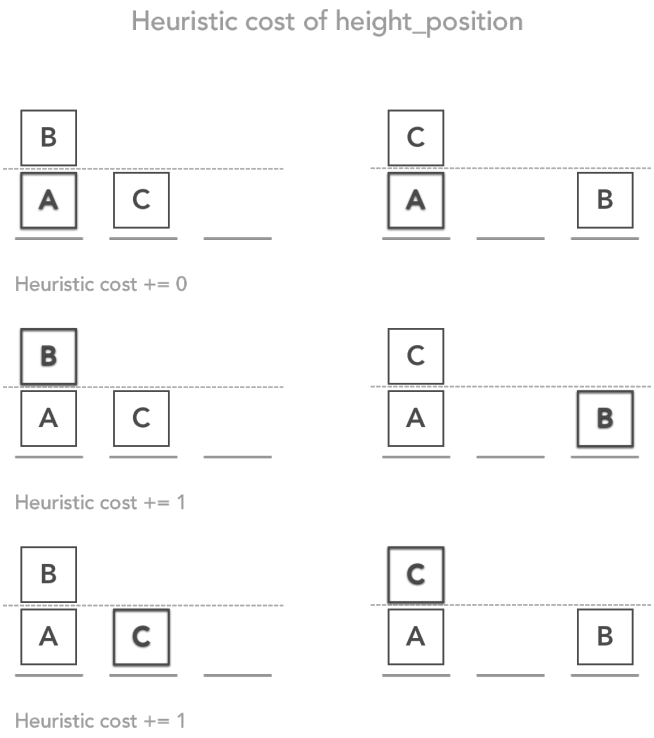
In this example we have the following Current and Goal states

So in the beginning of my heuristic function the heuristic cost is 0, but it will be incrementing first by the calculations based on **stack_index**:



Notice how the heuristic cost increments depending on the difference of stack index from elements in current state with elements from goal state.

Then, we also add the calculations of the second part of my heuristic, **height_index**:



Notice how the heuristic cost increments depending on the difference of height index from elements in current state with elements from goal state.

So for this current state the cost obtained from my heuristic would be:

h(state) = 3 + 2 = ***5***

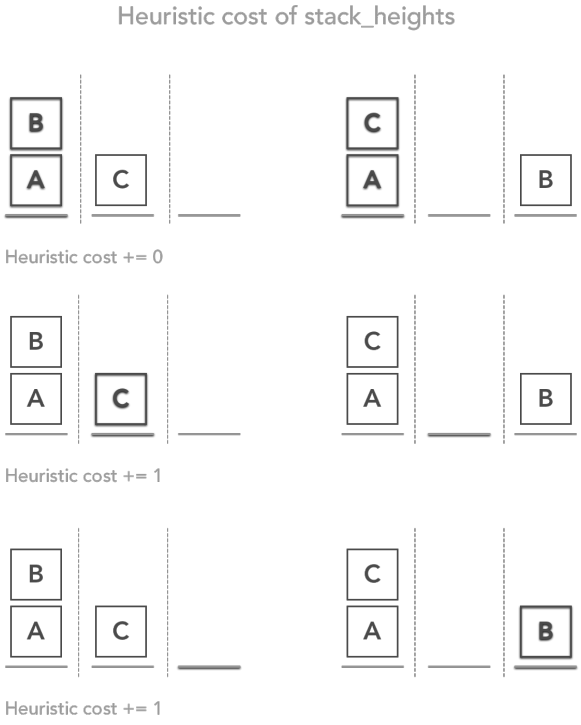
Note: In the case that the goal state contains X (do not care about the contents of that stack) I check if the element is important in the goal state (it is contained in a stack different than X), if not then I calculate the heuristic cost it would take me to move the element to a closer X stack.

Inconsistent Heuristic

For the inconsistent heuristic, I decided to use a calculation comparing the stack heights of the current state with the goal state. In other words, for every stack I compare its height with the same stack on the goal state, if it's different, I add the *abs* difference to my heuristic cost.

Using the same example as above:

My inconsistent heuristic cost calculation would be obtained like this:



Notice how the cost is the sum of all stack height differences between the current and final state.
In this example the heuristic cost would be:

h(state) = 1 + 1 = ***2***

Algorithms Analysis

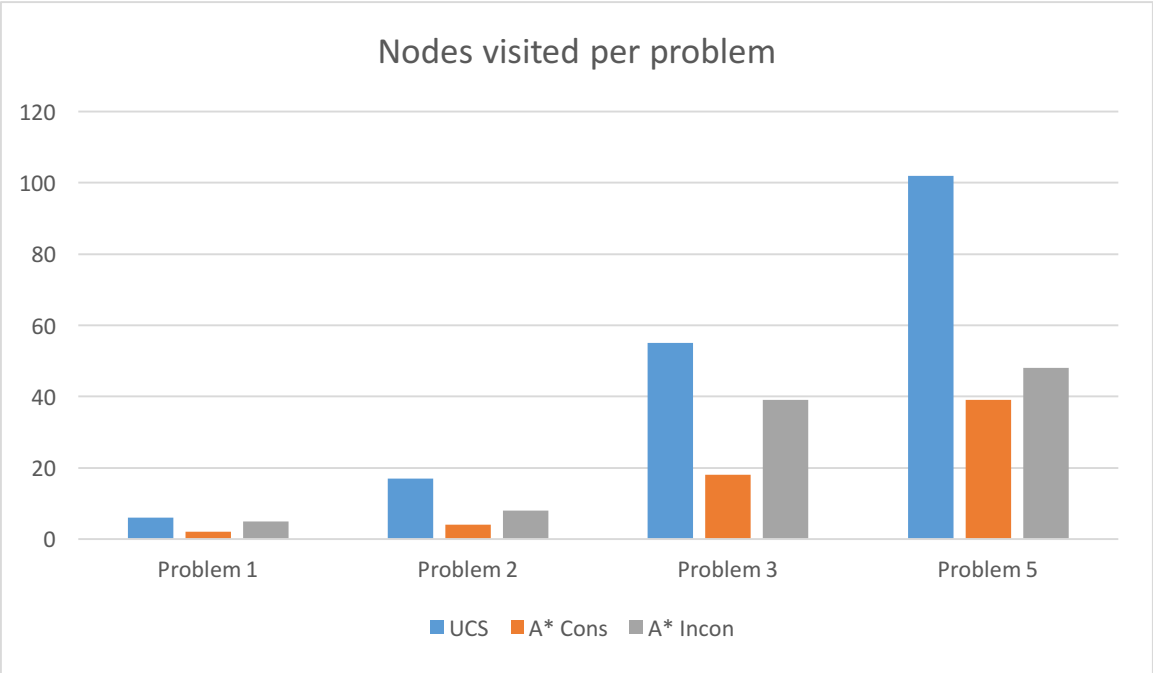
I decided to run my three algorithms with 9 different problems to check the efficiency and compare results. The three algorithms came up with the same optimal result, the main difference was in the efficiency on how many nodes were visited by each one.
The results of *how many nodes were visited* are presented in the following table:

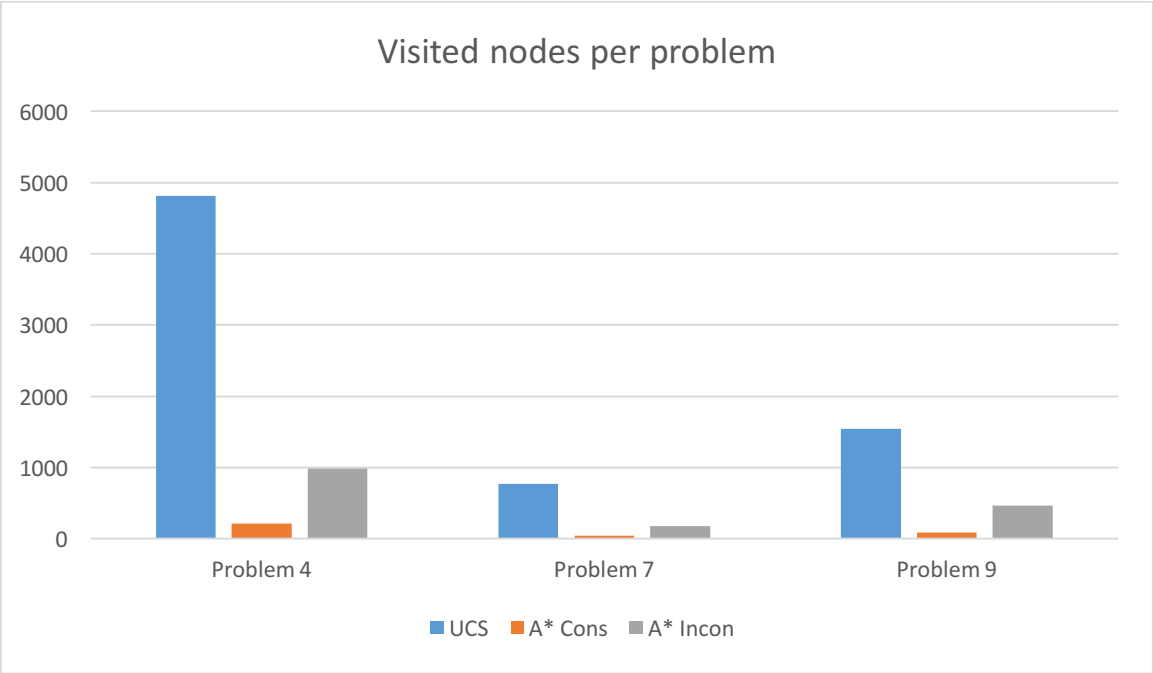
Note: All problems had the stack height limit set to 5.

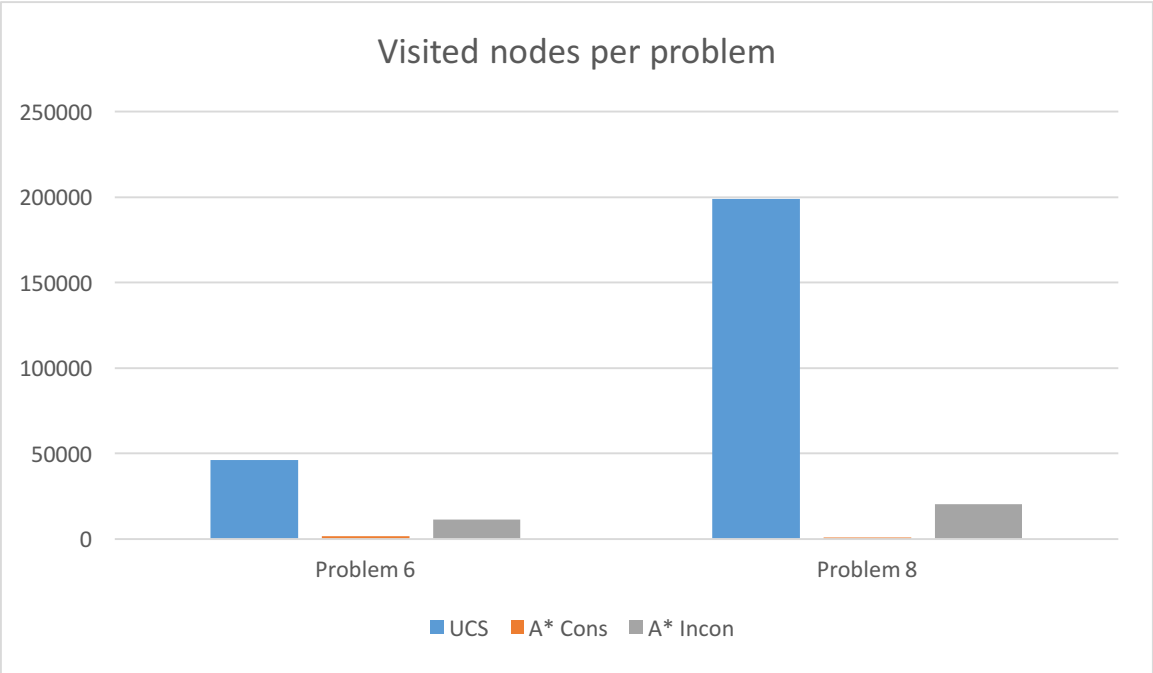
	Initial State	Goal State	UCS	A* Consistent	A* Inconsistent
1	'(A);(B);(C)'	'(A,C);X;X'	6	2	5
2	'(A);(B);(C)'	'(A,C);X;(B)'	17	4	8
3	'(A);(B);(C)'	'(C, A);X;(B)'	55	18	39
4	'(B, A); (C, D, E); ()'	'(A, E); (B); (D, C)'	4811	213	984
5	'(X, Y, Z); (); ()'	'(); (X, Y, Z); ();'	102	39	48
6	'(X, Y, Z); (); (A, B, C)'	'(A, B, C); (X, Y, Z); ();'	46359	1716	11274
7	'(A, B); (C, D); (); ()'	'(A, C); (B, D); (); ()'	767	40	180
8	'(A, B, C); (D, E, F); (); ()'	'(A, C, E); (B, D, F); (); ()'	198950	905	20571
9	'(A, B, C); (D, E, F); ()'	'(A, C, E, F); X; X;'	1540	85	463

As you can notice on the table my algorithm A* with consistent heuristic is by far the most efficient, the UCS is the least efficient. As I mentioned, the three output the same result but we can notice a huge improvement on efficiency with the A* algorithm specifically with a consistent heuristic.

The following graphs are a different representation of this analysis. You can notice the improvement more visual:


Nodes visited per problem
120
100
80
60
40
20
0
Problem 1
Problem 2
Problem 3
Problem 5
UCS
A* Cons
A* Incon


Visited nodes per problem
6000
5000
4000
3000
2000
1000
0
Problem 4
Problem 7
Problem 9
UCS
A* Cons
A* Incon



This effect occurs because having a heuristic is information useful for the algorithm to consider in the decision of which node to expand first. UCS consider the cost up to a node and that's all, thus you can see the gigantic number of visited nodes in more complex problems. On the other side, having a heuristic gives the algorithm extra information to make a better decision. Finally, having a consistent heuristic allows the algorithm to reach the optimal path with the less steps (visited nodes) as possible because it doesn't have to expand not optimal nodes.

Which algorithms are optimal?

The three algorithms are optimal because all came up with the optimal solution for the problem, despite some of them take more time than others. We may say that both A* algorithms are optimal because they are based on UCS, and because of UCS optimality, A* algorithm should be optimal in theory. UCS consider the cost on every step getting for sure an optimal path (if it's solvable). In this lab we reaffirmed this.

Conclusion

In my opinion, more complex algorithms (in this case A* consistent heuristic) are way more effective and useful that simpler algorithms. You get the huge advantage of visiting the less number of nodes as possible, which in complex problems is essential due to time limits. I still believe UCS is a good approach because it's easy to understand and implement, however as you can see on the charts, the difference in efficiently is enormous. Despite it is a real challenge to come up with a consistent heuristic, A* is the best algorithm of the ones analyzed in this lab. Complex, but way more efficient.