CZ3005: Artificial Intelligent
Problem Solving Lab Exercise

Zhang Xinye U1622118K SSP1

April 16, 2018

Introduction

In this lab we need to explore searching and CSP algorithms using the AISpace, and answer relevant questions.

Question One

(a) Give a graph where depth-first search (DFS) is much more efficient (expands fewer nodes) than breadth-first search (BFS).

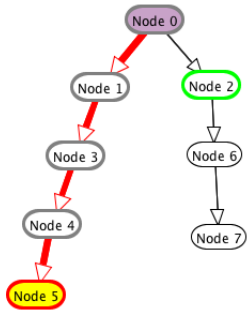

Figure 1: Question 1a

Search order is from *left to right*; Start node *Node 0*; goal node is *Node 5*.

(b) Give a graph where BFS is much more efficient than DFS.

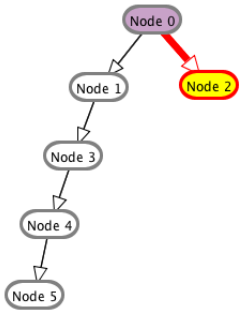

Figure 2: Question 1b

Search order is from *left to right*; Start node *Node 0*; goal node is *Node 2*.

(c) Give a graph where A* search is much more efficient than DFS or BFS.

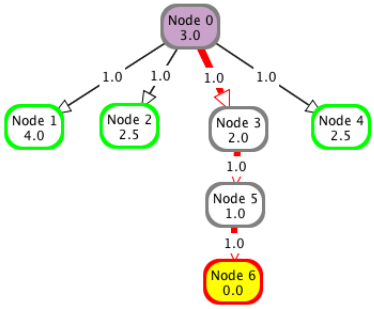

Figure 3: Question 1c

Search order is from *left to right*; Start node *Node 0*; goal node is *Node 6*.

(d) Give a graph where DFS and BFS is much more efficient than A* search.

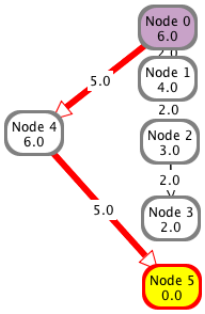

Figure 4: Question 1d

Search order is from *left to right*; Start node *Node 0*; goal node is *Node 5*.

Question Two

(a) Draw the graph in AIspace.org as a CSP problem.

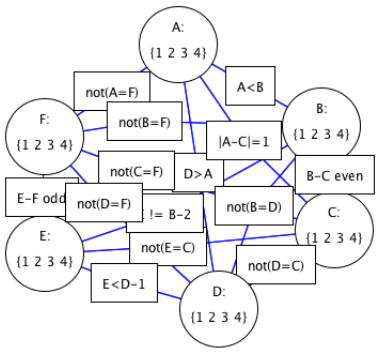

Figure 5: Question 2a

(b) For the first 5 instances of arc consistency, show which elements of a domain are deleted at each step, and which arc is responsible for removing the element.

The following listing shows how the first 5 domain s are reduced, and the corresponding arc responsible for them.

```
1 1 removed from the domain of B because of arc ( B, A<B )
```

```
4 removed from the domain of A because of arc ( A, A<B )
Arc ( C, |A-C|=1 ) is consistent
Arc ( A, |A-C|=1 ) is consistent
Arc ( C, B-C even ) is consistent
Arc ( B, B-C even ) is consistent
Arc ( D, not(B=D) ) is consistent
Arc ( B, not(B=D) ) is consistent
Arc ( A, D>A ) is consistent
1 removed from the domain of D because of arc ( D, D>A )
Arc ( B, not(B=D) ) is consistent
Arc ( C, not(D=C) ) is consistent
Arc ( D, not(D=C) ) is consistent
Arc ( C, not(E=C) ) is consistent
Arc ( E, not(E=C) ) is consistent
2 removed from the domain of D because of arc ( D, E<D-1 )
Arc ( B, not(B=D) ) is consistent
Arc ( A, D>A ) is consistent
Arc ( C, not(D=C) ) is consistent
4 removed from the domain of E because of arc ( E, E<D-1 )
...
```

(c) Show explicitly the constraint graph after are consistency has stopped.

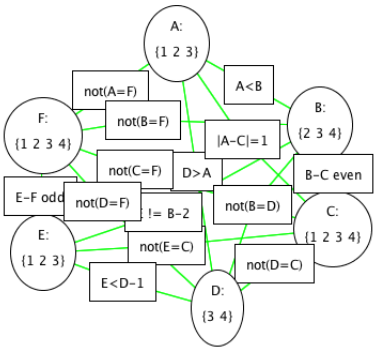

Figure 6: Question 2c

(d) Show how splitting domains can be used to solve this problem. Draw the tree of splits and show the solutions.

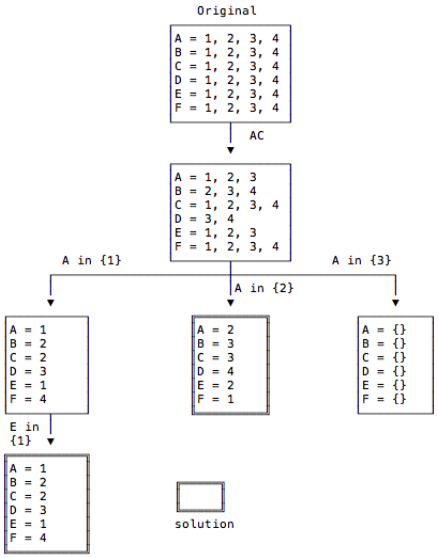

Figure 7: Question 2d

Question Three

(a) What is the effect of reducing *h(n)* when *h(n)* is already an underestimate?

If we reduce *h(n)* when *h(n)* is already an underestimate, the accuracy will not be effected(still optimal), but efficiency will decrease(expanding more nodes).

The intuitive idea of A* search is that a possible optimal path will "look good" and consequently be expanded first. On the other hand, if there is an alternative path that "looks better", it will be expanded earlier. More precisely, a path that "looks better" will have a lower "overall cost" *cost(n)+h(n)*, and *cost(n)* is the current cost, *h(n)* is the expected cost in the future.

Back to the question, if *h(n)* is an underestimate, alternative paths will looks better than they should have been, so that they would be expanded deeper than they should have been, which, in other words, expands more nodes.

In the following example, we compare two *h(n)*: Euclidean distance which is an underestimate, and half of Euclidean distance which is always smaller that the previous *h(n)*.

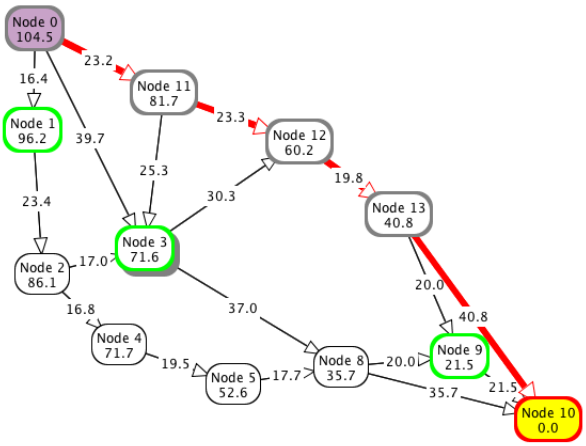

Figure 8: Question 3a: *h(n)* = Euclidean distance

Expands: 5 nodes Result:1,11,12,13,10

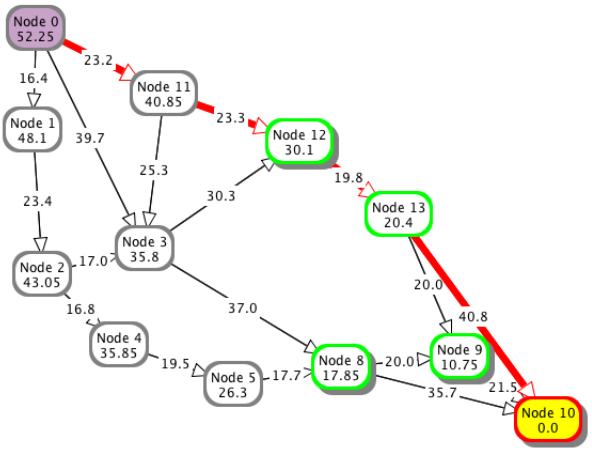

Figure 9: Question 3a: *h(n)* = half Euclidean distance

Expands: 16 nodes Result:1,11,12,13,10

We could see that the result are the same which is the optimal solution. However, the reduced *h(n)* has much larger expands that larger one.

(b) How does A* search perform when *h(n)* is the exact distance from n to the goal?

If *h(n)* is the exact distance from n to the goal, the accuracy will not be optimal, but efficiency will be optimal(expands fewest nodes among all h(n)).

Under such assumption where *h(n)* is the exact distance from n to the goal, *cost(n)+h(n)* will remain the same along the shortest path. The reason for that is for any node i in the shortest path form S to G, cost(i) is the smallest among all path from S to i, and h(n) is the smallest among all path from i to D. Therefore, all the frontiers that will be expanded will be on the shortest path form S to D, which means the efficiency is optimal.

The following example shows the case where *h(n)* is the exact distance from n to the goal.

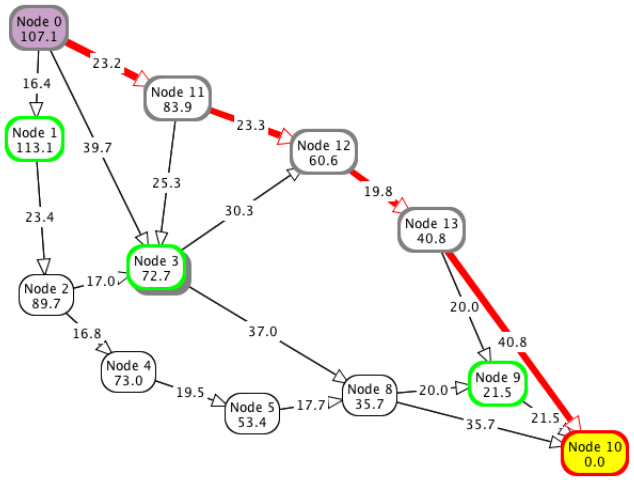

Figure 10: Question 3a: *h(n)* = exact distance to Goal

Expands: 5 nodes Result:1,11,12,13,10

(c) What happens if *h(n)* is not an underestimate?

If *h(n)* is not an underestimate, it could be exact distance from n to the goal, or an overestimate. The first case has already be discussed in 3.(b).

If *h(n)* is an overestimate, the solution may not be an optimal, the efficiency may be worse that other *h(n)*, however the result would largely depends on the problem itself. The reason for that is an overestimate will spoil the procedure of determining whether to stop exploring a path that may potentially be bad.

Question Four

Coming up will all the cases, and making all the graphs took about 3 hours, because of the limited functionality of AISpace.org. I suggest that a better tool for making and modifying the graph should be provided so that we could focus more on solving the problem itself. The overall time taken is about 5 hours.