CPSC 322 2018W2 Assignment 1

Make sure you follow all assignment instructions on the course website and in the question description on Canvas. Failure to do so will result in heavy penalties.

Make sure that in your answers you clearly indicate the exact section you are answering.

Please start each question on a new page (eg. don't put your answer to Q3 on the same page as your answer to Q2).

Question 1 (27 points): Comparing Search Algorithms

Consider the problem of finding the shortest path from *a* to *z* in the following directed graph:

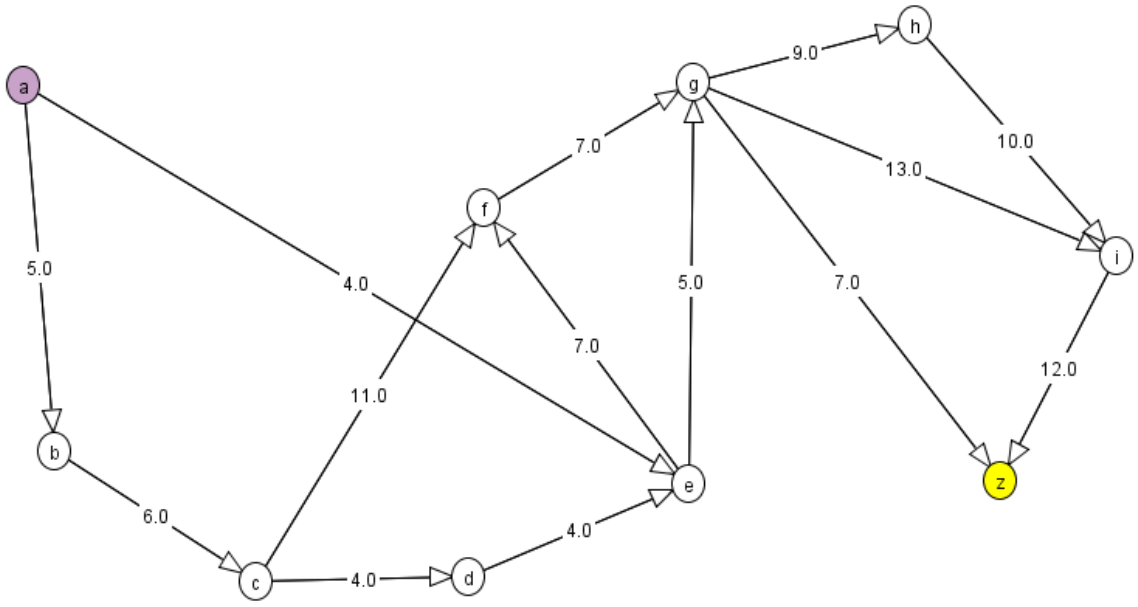


Figure 1. Graph for question 1. The arcs are labeled with their costs.

Start at node **a** and use **z** as the goal node. **Expand neighbours of a node in alphabetical order**, and **break ties in alphabetical order** as well. For example, suppose you are running an algorithm that does not consider costs, and you expand **a**; you will add the paths <*a*,*b*> and <*a*,*e*> to the frontier in such a way that <*a*,*b*> is expanded before <*a*,*e*>.

Additional notes:

- A node is expanded when the algorithm checks to see if it is a goal node, then returns the path if it is (or adds its neighbor paths to the frontier if it is not).
- If a given node is expanded multiple times, it should appear in your list of expanded nodes multiple times.

For the algorithms in questions 1.1-1.4, answer the following questions:

(a) What nodes are expanded by the algorithm? Order the nodes from first expanded to the last.
(b) What path is returned by the algorithm?
(c) What is the cost of this path?

1.1. **[5 points]** Depth-first search
1.2. **[5 points]** Breadth-first search
1.3. **[5 points]** A* (use the uninformative but valid heuristic $h(x) = 0$ for all nodes)
1.4. **[5 points]** Branch-and-bound (use $h(x) = 0$ for all nodes)

Answers:
DFS expands *abcdefghiz* and returns the path *abcdefghiz* which costs 64.
BFS expands *abecfgdfghiz* and returns the path *aegz* which costs 16.
A* expands *aebgcfdz* and returns the path *aegz* which costs 16. (A* with uninformative heuristic is equivalent to LCFS)
Branch-and-bound expands *abcdefghizizzghizfgefghzghiz* and returns the path *aegz* which costs 16.

1.5. **[7 points]**

(a) [1 point] Given your solutions in the previous questions, did BFS and Branch and Bound find the optimal solution?

(b) [3 points] Are BFS and Branch and Bound optimal in general? Explain your answer.

(c) [3 points] Did B&B expand fewer nodes than A*? Explain if your answer is true in general for these two algorithms and why.

Answer: BFS found an optimal solution this time but is not optimal (i.e. guaranteed to find an optimal solution) in all cases. This is because it is an uninformed search algorithm, blind to costs and heuristics. B&B found the optimal solution because it is optimal in general. Once it finds a solution and updates its bound, it continues to search but ignores paths that cost more than its updated bound. Therefore, if a better solution exists, we know that (1) it has not yet found it and (2) it will find it (or an even better solution). This means that, when the search is complete (i.e. there are no more paths with lower cost than the current bound), the most recently found solution must be an optimal one. B&B did not expand fewer nodes than A*. In general B&B cannot find a solution by expanding fewer nodes because A* is optimally efficient.

Question 2 (36 points): Uninformed Search: Peg Solitaire

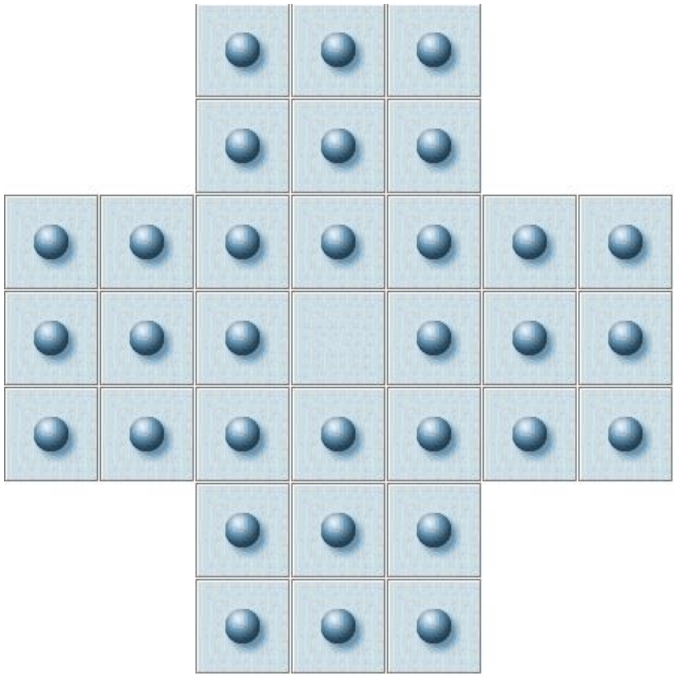


Figure 2. A sample Peg Solitaire board at the start of the game

Peg Solitaire is a board game for one player involving movement of pegs on a board with holes. The standard game fills the entire board with pegs except for the central hole. The objective is to empty the entire board except for a solitary peg in the central hole, by making valid moves. A valid move is to jump a peg orthogonally over an adjacent peg into a hole two positions away and then to remove the jumped peg.

From the Wikipedia entry: "Peg Solitaire"

You can play the game at http://www.novelgames.com/en/spgames/peg/

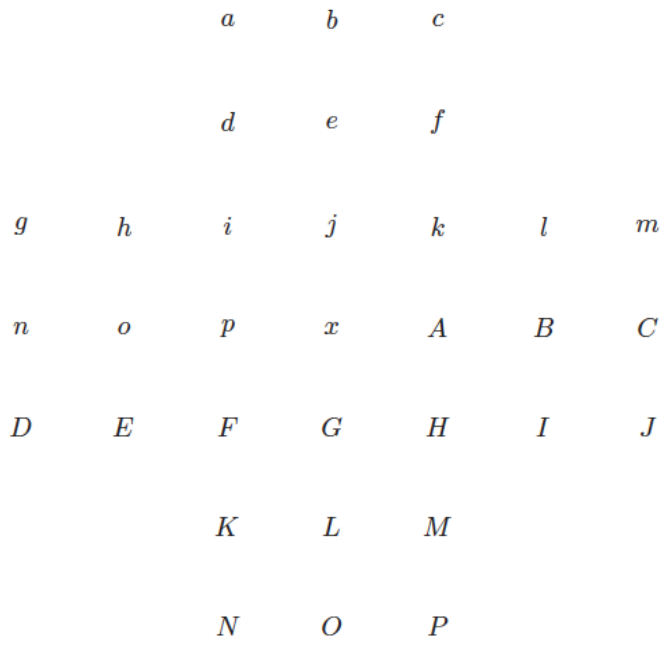


Figure 3. Peg Solitaire Board

2.1. **[12 points]** Represent peg solitaire as a search problem. (Use the labels provided in figure 3 for referring to spaces on the board).

(a) **[4 points]** How would you represent a node/state?
(b) **[2 points]** In your representation, what is the goal node?
(c) **[3 points]** How would you represent the arcs?
(d) **[3 points]** How many **possible** board states are there? **Note:** this is **not** the same as the number of "valid" or "reachable" game states, which is a much more challenging problem.

Answer:

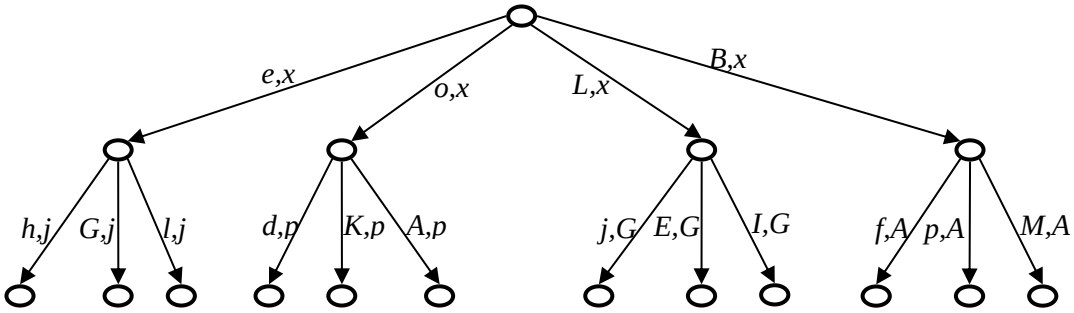
- **Node:** states representing which holes contain pegs and which do not. For example, we will use a sequence of 33 binary variables, one per hole.
- **Goal-nodes:** the node where only the variable representing hole *x* is true.
- **Arcs:** any valid move, represented by a 2-tuple of hole indexes, stating which holes the peg was moved from and to. For example, (*i*, *k*) means move the peg from *i* to *k* removing the peg in the hole between them (*j*).
- **Number of sates:** There are 2^{33} states

2.2. **[12 points]** The search tree:

(a) **[9 points]** Write out the first three levels (counting the root as level 1) of the search tree based on the labels in Figure 3. (Only label the arcs; labeling the nodes would be too much work).
(b) **[3 points]** What can you say about the length of the solution(s)?

Answer:

Given that there are 32 pegs, and we can remove only one peg per move, we will need 31 moves to remove the 31 pegs. Therefore the solutions are of length 31 (or 32 if counting length in terms of nodes/states rather than arcs).



2.3. **[12 points]** The search algorithm:

(a) **[4 points]** What kind of search algorithm would you use for this problem? Justify your answer.

(b) **[4 points]** Would you use cycle-checking? Justify your answer.

(c) **[4 points]** Would you use multiple-path-pruning? Justify your answer.

Answer:

Because this is an uninformed search problem, the only options are breadth-first search, depth-first search, and iterative deepening.

If we consider the total number of possible configurations (∼8 billion) then breadth-first search appears likely to run out of memory. However, (according to Wikipedia, anyways), the widest level of the tree is only ∼3 million nodes, making the problem more feasible.

Because we know that all solutions must exist at the same (32*nd*) level, fully exploring the earlier levels has no benefit.

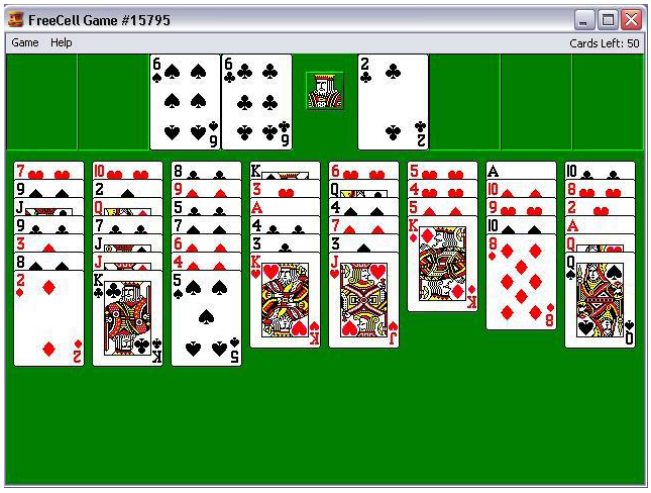
Cycles are not possible in this setting because every move removes a peg, making it impossible to get back to a state you were previously in. Therefore, there is no benefit to cycle checking.

Because there are no cycles and the tree has finite depth, depth-first search will be complete and have no risk of memory issues.

Multiple paths to the same state are possible (for example, any full valid solution could be rotated 90, 180 or 270 degrees). However, multiple path pruning could potentially run into the same memory problems as breadth-first search.

Question 3 (24 Points) Free Cell

Consider the problem of trying to solve a game of FreeCell in the minimum number of moves. You can play the game at http://www.freecell-cardgame.com/.
FreeCell comes standard on nearly every version of Microsoft Windows (including Windows 10, under the Microsoft Solitaire Collection), and full rules are also available at the Wikipedia entry:
https://en.wikipedia.org/wiki/FreeCell



The object of FreeCell is to move all the cards to the home cells [foundations], using the free cells as placeholders. To win, make four stacks of cards on the home cells [foundations], one for each suit, stacked in order of rank, from lowest (ace) to highest (king).

- *When moving cards to columns, cards must be moved in order from highest (king) to lowest (ace), alternating suit colors.*
- *When moving cards to home cells [foundations], cards must be moved in order from lowest (ace) to highest (king), same suit.*
- *A card from the bottom of a column can move to a free cell, the bottom of another column, or a home cell [foundation].*
- *A card from a free cell can move to the bottom of a column, or to a home cell [foundation].*

Microsoft Windows FreeCell help file

3.1. **[15 points]** Represent this as a search problem.
(a) **[7 points]** How would you represent a node/state?
(b) **[3 points]** In your representation, what is the goal node?
(c) [5 points] How would you represent the arcs?

Answer:

- Node: for example, a 16-tuple of vectors representing the 8 columns, 4 free-cells and 4 foundations.
- Goal-node: any state where every vector except the foundations is empty. And the cards in each foundation are of the same suit and ordered.
- Arcs: any valid move of the top card of vector v1 to vector v2, represented as a tuple (v1, v2). For example, (2, 12) means move the top card from vector 2 (column 2) to the top of vector 12 (free-cell 4).

3.2. **[9 points]** Give an admissible heuristic; explain why your heuristic is admissible. More points will be given for tighter lower bounds; for example, h=0 is a trivial (and useless) heuristic, thus it is not acceptable.
Answer:

- The number of cards not already on a foundation. This is admissible because every card will require at least one step to move it to a foundation.
- The number of cards not already on a foundation, plus one for every out of order card in the same

suit on the same column, e.g. the column [♥*K*, ♥*Q*, ♥*J*] requires at least five moves: two to remove the king and queen before the jack can move and three to move the cards to the foundations.

Question 4 (15 points) Modified Heuristics

To answer this question, the AISpace search applet will be useful. It can be found at
http://aispace.org/search/
For this question you are to think about the effect of heuristic accuracy on *A** search. That is, you are to experiment using a sample graph, and think about how the closeness of *h(n)* to the actual distance from node *n* to a goal affects the efficiency and accuracy of *A**.
Use the sample graph- Vancouver Neighbourhood Graph- in AISpace as an example. Set *h(n)* as an underestimate. To achieve this, you should first set the cost values automatically by clicking on the option '*Set edge costs automatically*' and then set the h(n) values to an underestimate by clicking on the option '*Set node heuristic automatically*' in the '*Search Options*' menu of the applet (***you need to be in CREATE mode in order to do this***). See how *A** works with this setting and report the number of nodes expanded as well as the optimal path found.
Then you will need to experiment with changing the *h(n)* values as described below. You can change the *h*(n) value of individual nodes by clicking on '*Set Properties'* in Create mode and then right click on the desired node. Or you can change the *h*(n) value globally by selecting '*set costs and heuristics*' under "*search options*". Remember not to modify the costs.
Notes:

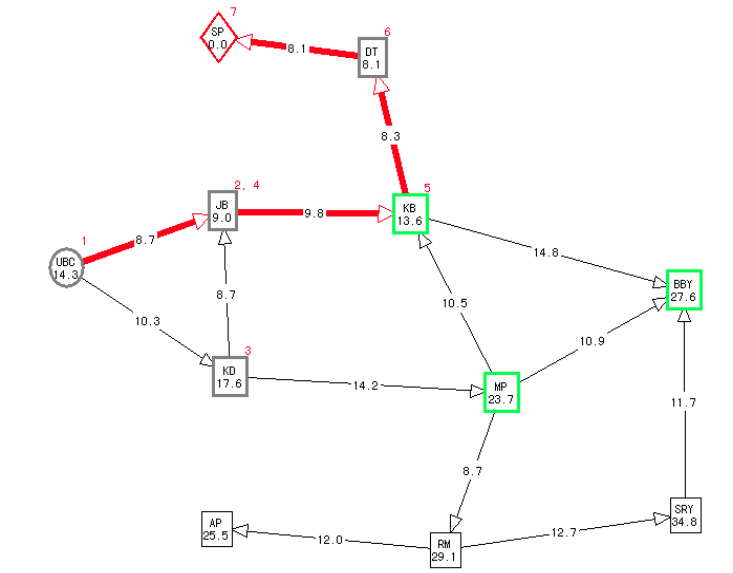
1. You need to set the cost values for this problem only once when you load the problem by selecting 'Set edge costs automatically' option. You will use these values throughout this question.
2. You need to reset the *h(n)* values to an underestimate using the '*Set node heuristic automatically*' as described above before each observation for this question.

4.1. [6 points 2+2+2] Reduce *h*(n), trying in three different ways:

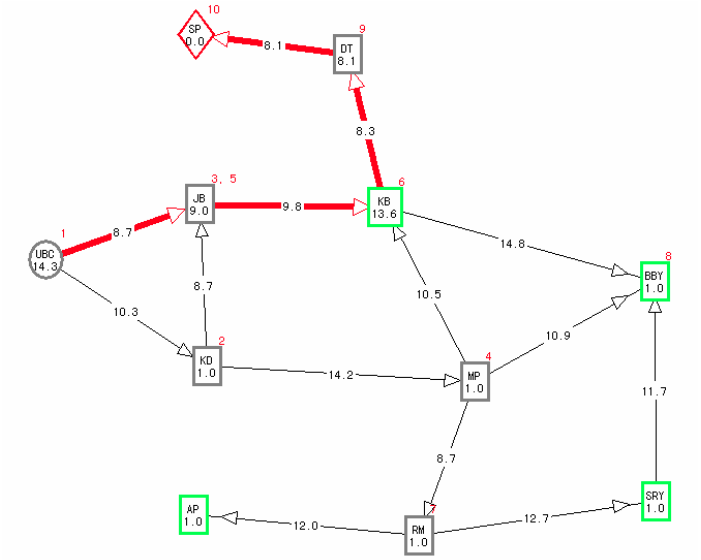
- First, subtract a large constant from all *h(n)*s; make sure all h values remain positive
- Second, only reduce *h(n)*s of the nodes on the solution path.
- Last, only reduce *h(n)*s of the nodes which are not on the solution path.
- *Note that the goal node is to be exempted from these changes, since its h(n) is already 0.*

Explain the effects of these changes, specifically:
(a) When can *A** still find the optimal path?
(b) When is the efficiency of *A** improved or reduced?
(c) Try to draw a more general conclusion regarding the changes in efficiency and optimality.

Answer:
Reducing h(n) when h(n) is already an underestimate won't affect the admissibility of A*; specifically, A* is still guaranteed to find a solution, and the first solution found is optimal. But reducing h(n) may reduce the efficiency of A*, expanding more nodes. More specifically, if all h(n)'s are reduced by the same value, it doesn't have any effect; if h(n) of those nodes which are not in the optimal path are reduced, the efficiency of A* will be reduced, while if h(n) of those nodes which are in the optimal solution path, the efficiency will improve.
Here's a Vancouver Neighborhood Graph, using A* search, the heuristic function is the Euclidean distance of the node to the goal node. A* expands 7 nodes. This is used as a comparison base.



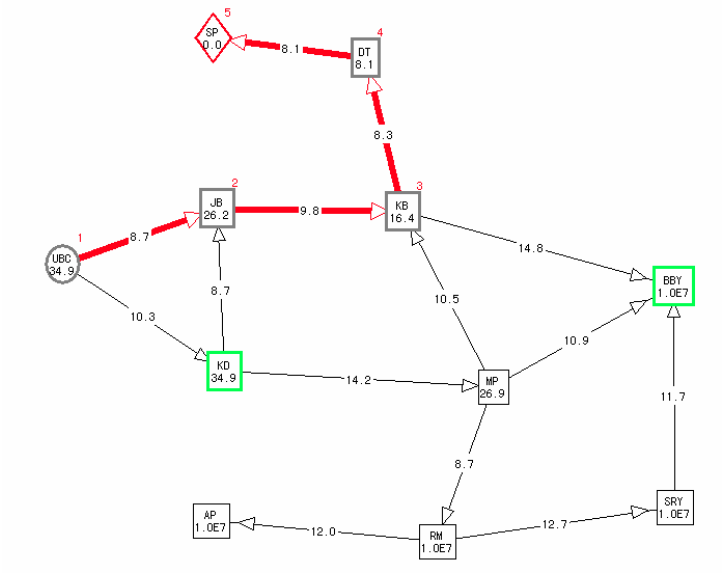
Below is the result after reducing the h(n) of all nodes not in the solution path to 1. Here A* expands 10 nodes, but the first solution found is still optimal.



4.2. [3 points 1+1+1] Set *h*(n) as the exact distance from n to a goal. To do this, you should add the costs of arcs on the optimal path from every node to the goal node, and set the sum as the heuristic function of the node.

(a) Can A* still find the optimal path?
(b) Is the efficiency of *A** increased or reduced?
(c) Try to draw a more general conclusion regarding the changes in efficiency and optimality.

Answer:
When h(n) is the exact distance from n to a goal, A* will find the solution directly, only expanding those nodes in the solution path, as shown below. A* only expands the 5 nodes in solution path, finding the optimal solution directly.



4.3. [6 points: 2+2+2] Increase *h*(n), also trying in three different ways.

- First, add a large constant to all *h(n)*s;
- Second, only increase *h(n)*s of the nodes on the solution path
- Last, only increase *h(n)*s of the nodes which are not on the solution path.

Discuss the effects of these changes, specifically:

(a) When can *A** still find the optimal path?

(b) When is the efficiency of *A** improved or reduced?

(c) Try to draw a more general conclusion regarding the changes in efficiency and optimality.

Answer:

If h(n) is not an underestimate, then the optimality of A* won't be guaranteed. When the h(n) of the nodes on the optimal solution path are say, much higher than the exact distance, the first solution A* finds may not be optimal. Below is an example: I set the h(n) of JB to 100, and A* first finds UBC->KD->MP->KB->DT->SP, not the optimal one:
UBC->JB->KB->DT->SP.