Computer Science

CompSci 367 S2 C
- ASSIGNMENT - TWO

The work done on this assignment must be your own work. Think carefully about any problems you come across, and try to solve them yourself before you ask anyone else for help. Under no circumstances should you work together with another student on any ***answers*** used in assignments.

Assessment

Due: Monday 24 September 2007 9am
Worth: 10%

Aim of the assignment

Warning: Read this entire document before writing to the forum about this assignment!!!

There are a plethora of search algorithms available. Ideally you should have already formed a rudimentary taxonomy of the ones we have looked at in this class. This taxonomy would distinguish between the algorithms along a number of design choice dimensions. The ones we are concerned with are:

1) The order that the search space is traversed (e.g., depth-first, breadth-first, etc.).
2) The cost metric being used (e.g., search depth, accumulated edge cost, etc.).
3) How duplicate states are handled (e.g., no checking, graph-based checking, etc.).
4) What information is being used (e.g., the problem statement, a heuristic evaluation function, etc.).
5) How nodes are selected to be expanded next (e.g., on the basis of the cost of the path from the root to the node, from the node to the cheapest goal, etc.).

These design choices affect the performance profile of the resulting search algorithm. The performance profile would include such things as:

- The space and time complexity of the algorithm.
- Its completeness.
- Its optimality.

You should understand the design choices and their performance consequences, as well as the assumptions underlying those performance consequences. For example, even in a finite search space, depth-first search is not complete if there are cycles in the space. The different design choices have arisen as responses to performance deficiencies in earlier algorithms' design choices. Hopefully, you will come to understand the set of existing search algorithms as a dance between performance problems and design choices. Where a perceived problem leads to a suggested solution which leads to new perceived problems and new suggested solutions, and so on.

This assignment will ask you to try to understand this dance. Some of the questions will be trivial and you will be able to just look them up. However, you should try to understand the why's and how's of even these answers. Other questions will require you to think a bit and perhaps to experiment some. There are also some questions which (while seeming straight forward) are unexpectedly difficult and may require systematic experimentation. To do well on this assignment you will need to ***think*** about what is going on with the different combinations of design choices. Oftentimes, the best way to think is to talk and/or write about the problem.

I encourage you to use the forum on this assignment. You can use the forum to discuss various aspects of the assignment:

- You are free to discuss whether a question is trivial or hard.
- You are free to discuss theoretical problems as long as you don't state which question it is relevant to (e.g., you might discuss whether duplicate checking affects the completeness).
- You are free to discuss what a question is really asking.
- You are free to discuss the graph search program supplied with this assignment (as long as you don't discuss any graph that might be submitted as part of this assignment).

The guiding principle underlying what is legitimate to discuss on the forum is whether it allows someone who doesn't **understand** the discussion to simply write down the answer. If it does then you shouldn't be saying it in the forum. The idea behind discussing it on the forum is to help you develop the theoretical understanding necessary to correctly answer the questions. In other words, the goal (as I see it which is obviously different than how some of you may see it :^) is to get you engaged in deepening your understanding, ***not*** to give you a chance to create a "cheat" sheet.

I'm hoping you will enjoy this exercise, so grab a cup of coffee (substitute beverage of choice) and try to figure out what's going on in these questions. Also, make sure you give yourself a *lot of time* for this assignment. If you leave it to the last moment, *you won't enjoy it!*

Marking Notes

For this assignment the search spaces have the following properties:

1) The search spaces are finite, though the search trees created by traversing these spaces may be infinite.
2) Edges in a space may have either uniform costs or varying costs.
3) There may be cycles in the search space.
4) The heuristic evaluation functions are both admissible and consistent.

This means there are four different types of search spaces. These four types of spaces come from the two possibilities available in properties 3) and 4).

The search algorithm design choices you will need to consider are:

1) The order that the search space is traversed (e.g., depth-first, breadth-first, etc.).
2) The cost metric being used (e.g., search depth, accumulated edge cost, etc.).
3) How duplicate states are handled (e.g., no checking, graph-based checking, etc.).
4) What information is being used (e.g., the problem statement, a heuristic evaluation function, etc.).
5) How nodes are selected to be expanded next (e.g., on the basis of the cost of the path from the root to the node, from the node to the cheapest goal, etc.).

The questions will partially specify a search algorithm. They will specify a primary search algorithm (e.g., depth-first search) and some variations (e.g., with and without duplicate (state) checking). The primary search algorithm normally has a default for each design choice and unless the variation modifies that choice it should be considered to be in effect (e.g., for depth-first search with/without duplicate checking, the cost metric default is search depth).

Each complete question will be worth the same (1 mark) regardless of whether it is trivially easy or diabolically hard. The answers to each part of the question can be *yes*, *no*, or *depends*. *Yes* means that regardless of the type of search space the answer is "yes" for that primary search algorithm with that variation. *No* means that regardless of the type of search space the answer is "no" for that primary search algorithm with that variation. *Depends* means that for some search spaces the answer is "yes" and for some it is "no".

If the answer is *yes*, then you need to **briefly** explain why it is yes. If the answer is *no*, then you need to give a **small** counter-example and explain how your example is a counter-example. If the answer is *depends*, then you need to explain which search space property it depends on and under which combination of property values it would be *yes* and why, and under which values it would be *no* and give a counter-example. The counter-example needs to be a search graph that can be run using the graph search program supplied with the assignment (you create the graph and then "save" it).

Note: You will probably want to use the supplied graph search program to explore your understanding of these search design choices and do your experiments.

Note: There are webpages and video files on how to use the graph search program.

Note: Iterative-deepening is not one of the search options available with this program. However, you can simulate it using the "user defined" search option.

Note: Search algorithms can have many different names and many different descriptions. For example, "graph search" can also be described as "searching with duplicate checking". One of your jobs in this assignment is to recognize these synonymous phrases. This is also something that can be discussed on the forum.

Questions:

Question 1.

a. What is space complexity of depth-first search with duplicate checking?

[.5] The space complexity of DFS with duplicate checking is exponential.

[.5] It is exponential because the table used for duplicate checking records each new state created and the number of states created grows exponentially with respect to the length of the longest explored path.

b. What is space complexity of depth-first search without duplicate checking?

[.5] The space complexity of DFS without duplicate checking is linear.

[.5] It is linear because all that is stored is the current path.

Question 2.

a. Is depth-first search complete with duplicate checking?

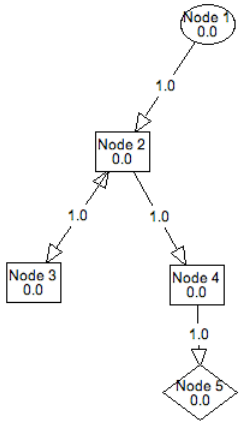
[.5] Yes, DFS is complete with duplicate checking.

[.5] It is complete because the number of states is finite and duplicate checking prevents us from expanding nodes that have a state associated with a node that has already been expanded. Given a finite search graph, DFS will eventually find a goal node (assuming there is one).

b. Is depth-first search complete without duplicate checking?

[.5] No, DFS is not complete without duplicate checking.

[.5] Example is Ex2b.xml (see figure on next page)


Node 1
0.0
1.0
Node 2
0.0
1.0
1.0
Node 3
0.0
Node 4
0.0
1.0
Node 5
0.0

Question 3.

a. What is the space complexity of iterative-deepening with duplicate checking?

[.5] The space complexity of IDS with duplicate checking is exponential.

[.5] It is exponential because the table used for duplicate checking records each new state created and the number of states created grows exponentially with respect to the length of depth limit path.

b. What is the space complexity of iterative-deepening without duplicate checking?

[.5] The space complexity of IDS without duplicate checking is linear.

[.5] It is linear because all that is stored is the current path.

Question 4.

a. Is iterative-deepening with duplicate checking, complete?

[.5] IDS with duplicate checking is complete.

[.5] It is complete because IDS searches all paths to the current depth limit depth and if it fails to find a goal node, then it increases the depth limit by one. When the depth limit equals the length of the optimal solution path, then that solution node will be found.

b. Is iterative-deepening without duplicate checking, complete?

[.5] Yes, IDS without duplicate checking is complete.

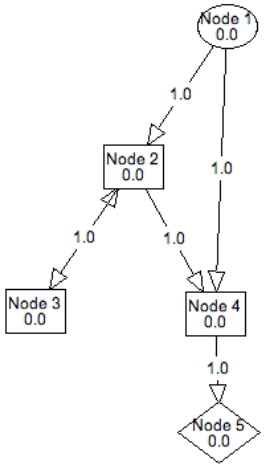
[.5] The reason is the same as for 4.a

Question 5.

a. Is iterative-deepening with duplicate checking, optimal?

[.5] No, IDS with duplicate checking is not optimal.

[.5] IDS with duplicate checking (as duplicate checking is shown on page 67 of lecture 1 on uninformed search) is not optimal because if, on the depth limit equal to the optimal path length, the search first encounters a state that is on the optimal path to the goal but it encounters it on a non-optimal path from the root then when later during that iteration it encounters that state the second time, it will not be expanded and so the optimal path will not be discovered. Look at file Ex5a, the optimal path is from the root (node 1) to Node 4 to the goal (Node 5) for a solution path length of 2. When the depth limit is set to 2, the IDS will first visit Node 4 via Node 2 (and not go on to Node 5 because that path has a length of 3). When IDS then revisits Node 4, it will be detected as a duplicate state and not expanded, so the optimal path will not be discovered. The path that will be discovered on the next iteration is Node 1 -> Node 2 -> Node 4 -> Node 5 (a non-optimal length of 3).



b. Is iterative-deepening without duplicate checking, optimal?

Yes, IDS without duplicate checking is optimal.

It is optimal because, now when the depth limit is the same as the optimal solution length, IDS will find the optimal solution path.

Question 6.

a. Is using breadth-first search with accumulated edge costs complete?

This is the same as uniform cost search and is complete.

It is complete because the assumptions are that there is some positive minimal value, e, for edge costs and that there are only a finite number of states, n. First, note for any given cost, c, there are only finitely many paths that have a lower or equal cost (namely, n ** (c/e)). Let c be the optimal solution path cost, since there are only finitely many paths that have a lower or equal cost, UCS will eventually find it.

b. Is using breadth-first search without accumulated edge costs complete?

Yes, BFS without accumulated edge costs is complete.

BFS is just a special case of UCS where depth can be thought of as having each edge have a cost of 1, therefore the same argument as above applies here.

Question 7.

a. Is using breadth-first search with accumulated edge costs optimal?

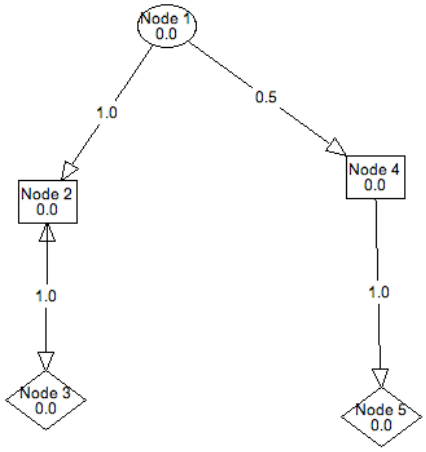
Yes, UCS (the standard name for this algorithm) is optimal.

UCS is optimal because we explore the paths in order of increasing costs, therefore the first goal node we select to expand will be on the optimal path.

b. Is using breadth-first search without accumulated edge costs optimal?

No, BFS without accumulated edge costs is not optimal.

Example can be seen in file 7b.xml. Goal Node 3 will be found before goal Node 5.



Question 8.

a. Is using uniform-cost search, when it selects the node to expand next based on the distance from the root to that node, complete?

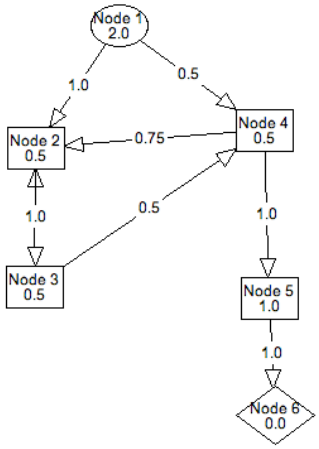
Yes, UCS (this is the standard one) is complete.

The argument here is the same as for question 6a.

b. Is using uniform-cost search, when it selects the node to expand next based on the estimated distance from that node to the nearest goal, complete?

This is the same as greedy search and is not complete.

The example for this is 8b.xml. There is a loop from Node 1 -> Node 2 -> Node 3 -> Node 4, because the heuristic value of each of these nodes is .5 while h(Node 5) = 1 {which is the node that leads to the goal}.



Question 9.

a. Is using uniform-cost search, when it selects the node to expand next based on the distance from the root to that node, optimal?

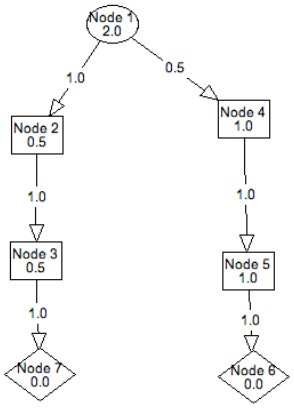
Yes, UCS is optimal.

The argument for this is the same as for 7a.

b. Is using uniform-cost search, when it selects the node to expand next based on the estimated distance from that node to the nearest goal, optimal?

This is the same as greedy search and is not optimal.

An example of a search tree where greedy search is not optimal is shown in file 9b.xml. Greedy search will find the path Node 1 -> Node 2 -> Node 3 -> Node 7.



Question 10.

a. Is using uniform-cost search, when it selects the node to expand next based on the estimated distance from that node to the nearest goal, optimally efficient?

This is also known as greedy search and is not optimally efficient.

Since it is not guaranteed to find optimal solutions, it cannot be optimally efficient.

b. Is using uniform-cost search, when it selects the node to expand next based on the sum of the distance from the root to that node and the distance from the root to that node, optimally efficient?

This is also known as A* and is optimally efficient.

To show that A* is optimally efficient we assume that it's not and show that leads to a contradiction. Assume A* is not optimally efficient, then there is some non-goal node, n, that A* expands but which some other less informed optimal algorithm, X, does not expand. Since X is less informed than A*, then $h_{A^*}(n) > h_X(n)$ ***and therefore*** $f_{A^*}(n) > f_X(n)$***. Since A* expanded n,*** $f_{A^*}(n) =< f^*(n)$ ***and therefore*** $f_X(n) < f^*(n)$***. Consequently, X should have expanded n if X is guaranteed to be optimal. This a contradiction, therefore X cannot be optimal and less informed and not expand n. Therefore A* is optimally efficient.***

Submission:

Use an appropriate editor to enter your answers onto this form. Let me know as soon as possible if there are any problems.
Submit the following:

1. This form with your answers.
2. The graph search files (named <question number>.xml, e.g., "13a.xml" for a counter-example search graph for question 13a) used to demonstrate counter-examples.

Submit these files to the electronic drop box.