15-381 Spring 06 Assignment 1: Search

Questions to Sonia Chernova (soniac@cs.cmu.edu)
Out: 1/24/06 Due: 2/7/06

January 26, 2006

Please turn in your answers on this assignment (extra copies can be obtained from the class web page). This written portion must be turned in at the beginning of class at 1:30 on February 7th. The code portion must be submitted electronically by 1:30pm on February 7th. Please write your name and Andrew ID in the space provided on the first page, and write your Andrew ID in the space provided on each subsequent page. This is worth 5 points: if you do not write your name/Andrew ID in every space provided, you will lose 5 points.

Code submission. To submit your code, please copy all of the necessary files to the following directory:

/afs/andrew.cmu.edu/course/15/381/hw1_submit_directory/yourandrewid

replacing yourandrewid with your Andrew ID. You can use any of the following programming languages: C/C++, Java, Perl, Matlab, Lisp, or Python. If you would like to use a language that is not on this list, please check with us first. All code will be tested on a Linux system, we will not accept Windows binaries. No matter what language you use, you must ensure that the code compiles and runs in the afs submission directory. Clearly document your program.

Late policy. Both your written work and code are due at 1:30pm on 2/7. Submitting your work late will affect its score as follows:

- If you submit it after 1:30pm on 2/7 but before 1:30pm on 2/8, it will receive 90% of its score.
- If you submit it after 1:30pm on 2/8 but before 1:30pm on 2/9, it will receive 50% of its score.
- If you submit it after 1:30pm on 2/9, it will receive no score.

Collaboration policy. You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas in the class in order to help each other answer homework questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers
- not to copy answers

- not to allow your answers to be copied

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we ask that you specifically record on the assignment the names of the people you were in discussion with (or "none" if you did not talk with anyone else). This is worth five points: for each problem, space has been provided for you to either write people's names or "none". If you leave any of these spaces blank, you will lose five points. This will help resolve the situation where a mistake in general discussion led to a replicated weird error among multiple solutions. This policy has been established in order to be fair to the rest of the students in the class. We will have a grading policy of watching for cheating and we will follow up if it is detected.

1 Formulating the Search Problem (15 points)

References (names of people I talked with regarding this problem or "none"):

The four-peg version of the Tower of Hanoi puzzle consists of four pegs mounted on a board and n disks of various sizes with holes in their centers (see Figure 1). If a disk is on a peg, only a disk of smaller diameter d can be placed on top of it. Given all the disks properly stacked on the leftmost peg as in Figure 1, the problem is to transfer the disks to the rightmost peg by moving one disk at a time. Formulate this problem as a search problem.

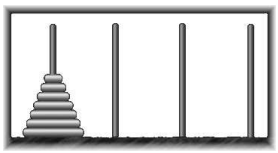


Figure 1: Four-peg version of the Tower of Hanoi.

1.1 (3 points)

Define a state representation.

1.2 (3 points)

Give the initial and goal states in this representation.

1.3 (3 points)

Define the successor function, i.e. show how the successors of a state can be computed using your representation.

1.4 (3 points)

What is the cost function for the above successor function?

1.5 (3 points)

What is the total number of legal states?

2 Uninformed Search (17 points)

References (names of people I talked with regarding this problem or "none"):

2.1 (9 points)

For each of the following statements, explain in terms of the cost function the circumstances under which it is true.

- Breadth First search is a special case of Uniform Cost search.

- Breadth First search is a special case of Best First search.

- Uniform Cost search is a special case of A* search.

2.2 (5 points)

Give a description of a search space in which Iterative Deepening performs much worse than Depth First search. Give the respective complexities for this domain in Big O notation.

2.3 (3 points)

Describe the criteria that determine the best direction (forward or backward) for search in a problem space. Assume a single start state and a single goal state.

3 Informed Search (18 points)

References (names of people I talked with regarding this problem or "none"):

3.1 (3 points)

Suppose that in addition to the admissible heuristic h(n), you are told that there is a solution whose true cost is K. How would you change A* to take advantage of this knowledge and reduce the number of nodes that need to be expanded? Does this method maintain optimality?

3.2 (3 points)

Under what conditions does Simulated Annealing perform better than Hill Climbing? Would you ever prefer Hill Climbing? If so, when?

3.3 (12 points)

In an environment that is an $n \times n$ grid, we have n cars that are located in squares (1,1) through (1,n) (i.e. in the bottom row). All of the cars have to be moved to the top row of the grid, but their order must be reversed. The car that started at (1,i) must be moved to (n,$n - i + 1$), and so on. During each timestep, every one of the cars executes a legal move at the same time. There are five legal moves: North, East, South, West or Stay. Assume that multiple cars can be located in the same grid square.

(a)(3 points) The size of the state space is

(i) $O(n^2)$ (ii) $O(n^3)$ (iii) $O(n^{2n})$ (iv) $O(n^{n^2})$

(b)(3 points) The branching factor is roughly

(i) 5 (ii) $5n$ (iii) 5^n

(c)(3 points) Suppose that car i is currently located at (x_i,y_i). Write a nontrivial admissible heuristic h_i for the number of moves it will take for that car to get to its goal location (n,$n - i + 1$), assuming that there are no other cars on the grid.

(d)(3 points) Take the problem of moving *all* of the n cars to their destination. Which of the following heuristics are admissible when considering all of the cars at the same time?

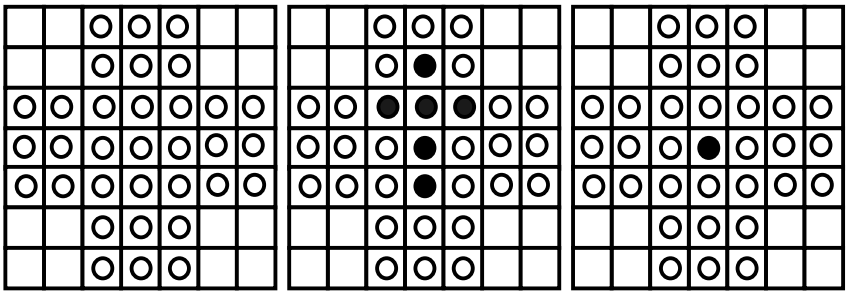
(i) $\Sigma_{i=1}^{n} h_i$ (ii) $max\{h_1, ...h_n\}$ (iii) $min\{h_1, ...h_n\}$ (iv) None of these

4 Peg Solitaire (50 points)

References (names of people I talked with regarding this problem or "none"):

Peg Solitaire is a game played on a board consisting of a number of holes and pegs. The object of the game is to make a series of *jumps* such that there is only 1 peg left, this peg must end in the center hole. A jump is when a peg is moved over an adjacent one and lands on an empty cell next to it. The peg that was jumped over is then removed from the board. The jumping sequence is similar to that of checkers except that instead of diagonal jumps, the jumps are made either horizontally or vertically.

Different variations of the game exist, with varying board sizes and shapes. In this assignment we will use one standard board shape with several starting peg configurations. The figure below shows the board, an example peg configuration and the goal state. To try out the game and see more examples go to http://www.puzzle-factory.com/java/32peg2/java-solitaire2.html.



(a) (b) (c)

Figure 2: (a) The empty board, (b) the "Latin cross" peg configuration, (c) final board. Squares with empty circles represent empty holes, full circles represent pegs.

Implement an algorithm that outputs a series of jumps for solving the puzzle given the initial board configuration as the input. Note that the goal is not only to have only one peg remaining, but also that it must be located in the center hole. Several example input and output files will be provided for testing.

Files describing the initial board configuration will visually represent the board using a string of characters. Each line of the file represents one row of the board. The following is a representation of example board (b) in the above figure:

```
--ooo--
--oxo--
ooxxxoo          -  flat space, not usable
oooxooo          o  hole
oooxooo          x  peg
--ooo--
--ooo--
```

Given the initial board configuration as input, your program should output the sequence of jumps needed to solve the puzzle. Number each of the holes according to the diagram below. Each jump should be described by the pairing $[initState, endState]$, where a peg starting at $initState$ jumps to location $finalState$ and the peg in the middle is removed. Here is the example output for the "Latin cross" puzzle:

[9, 7]
[23,9]
[10,8]
[7,9]
[4,16]

Note that many peg configurations have multiple solutions, including the one above.

		0	1	2		
		3	4	5		
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
		27	28	29		
		30	31	32		

Figure 3: Hole numbering.

4.1 (2 points)

How many different possible game states are there in this puzzle? (Define game state as at least one peg is on the board)

4.2 (2 points)

Define a state representation.

4.3 (6 points)

Give two admissible heuristics for this problem:

(a)

(b)

4.4 (10 points)

Implement Depth First Search to solve the Peg puzzle. In the table below, record the number of states visited by your algorithm and the effective branching factor for each of the puzzles. The effective branching factor is defined as follows: if during the search the algorithm expands N nodes, and the solution is found at depth d, then the effective branching factor b^* is the branching factor that a uniform tree of depth d would have to have in order to contain N nodes. More specifically the variables are related by the equation: $N = 1 + b^* + (b^*)^2 + ... + (b^*)^d$.

Puzzle Type	# Pegs	# States Visited	Branching Factor
Small House	7		
Fireplace	11		
Pyramid	16		
Empty Diamond	20		
Diamond	24		

4.5 (10 points)

Implement A* to solve the Peg puzzle. In the table below, record the number of states visited by your algorithm and the effective branching factor for each of the puzzles.

Puzzle Type	# Pegs	# States Visited	Branching Factor	Max Queue Size
Small House	7			
Fireplace	11			
Pyramid	16			
Empty Diamond	20			
Diamond	24			

Describe the heuristic you are using for A*.

4.6 (3 points)

How do you expect the performance of IDS to compare to DFS in this domain?

4.7 (3 points)

Would Bi-Directional search be useful in this domain?

4.8 (14 points)

In addition to your code, please also submit a brief (about one page) description of your algorithms. This must be attached to the rest of the written part of your assignment. The report should include a description of your implementation and a brief discussion comparing the performance of DFS and A* in this domain.

4.9 Code Submission Details

Program Output Your program should output the number of expanded states, the average branching factor, and the sequence of jumps in the solution. Use the provided output files as examples of the output format. Note that we provide

one example solution files per puzzle, even though most puzzles have multiple possible solutions. Your solution does not have to match the one in the file, it just has to correctly solve the puzzle using the legal steps.

Documentation A README file must be submitted with the source code that specifies how to compile and run your code. Specifically the file must describe the command line arguments needed to run the different search algorithms and different puzzle files.

Grading Grades will be assigned based on the correctness of the solution output by the program. All programs will be evaluated using the same set of puzzles. While some of the puzzles in the evaluation set will be from the sample set of files provided with the homework, additional puzzles from the game website will be used. We encourage you to generate additional board input files for your own testing.

4.10 Extra Credit (20 points)

It is obvious that even when multiple solutions to a puzzle exist, the number of jumps required to win on a board of n pegs is n-1. As a result, people have come up with another metric for evaluating the solutions to this type of puzzle. Let us call a sequence of consecutive jumps made by the same peg a single *move*. For example $[9, 7][7, 21][21, 23]$ is all one move because the initial state of the second and third jump is equal to the final state of the previous one, so the same peg is used to perform this entire sequence.

Modify your algorithm to return the solution with the smallest number of moves. Some small puzzles to try are Small House, Small F, Easy, and Heart. All of these and more can be found at http://www.puzzle-factory.com/java/32peg2/java-solitaire2.html, which also lists the minimum number of moves needed to solve each puzzle. Be sure to include a thorough description of your approach with the other algorithm descriptions.