

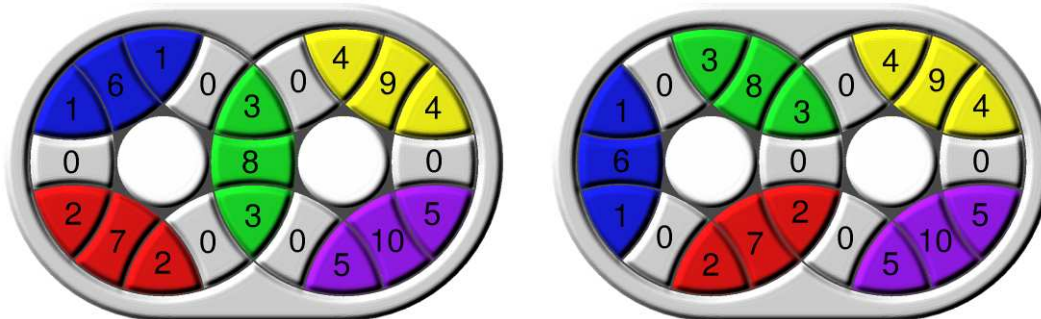
CSIS0270 Artificial Intelligence, 2003–2004

Assignment 1

Deadline Feb 20, 2004, 5:00pm.

This assignment contains a written part and some programming parts. For the written part, hand-in your answers to the assignment box (B3). For the programming part, hand-in the source code of the programs that you write via the hand-in system of the department. You may also type your answer and send it to us using the handin system even for the written parts, but don't use a word document. Instead, just send us a plain text file that contains your answer.

- Repeated states handling (10%)** As is discussed during the lecture, if we run the *GraphSearch* algorithm in the lecture notes when performing a uniform cost search, it is possible to have repeated states in the priority queue, although they will not be expanded multiple times.
 - If we are performing BFS, it is easy to make sure that no states will be in the queue twice. Write the exact algorithm, without using any more data structure than those that are already in the algorithm.
 - Modify uniform cost search so that, at the same time, there will never be a repeated state in the queue. You should maintain optimality and efficiency of uniform cost search. Specify all the data structures that you use.
- Uninformed search algorithms (35%)** A puzzle is played with a small gadget that looks like the following (the numbers are added for denoting different blocks in the gadget, they are not present in the real gadget).



The gadget contains two wheels of colored blocks, some of the blocks are shared by the two wheels. There is a natural placement of the blocks, which is shown in the figure on the left. The objective of the game is that given an initial placement, you would like to rotate the wheels so that the natural placement is achieved. Each time you can rotate either wheels in either direction for one "step". E.g., the configuration on the right is achieved from the natural configuration by rotating the left wheel counter-clockwise for one step.

Write three programs, to solve the problem using (1) BFS, (2) IDS and (3) BDS, avoiding repeated states just by not going back to the state that you come from. The input of the program should be a configuration, encoded using the following scheme:

- First list the blocks in the left wheel in counter-clockwise direction, from the block next to the shared blocks. E.g., for the natural configuration, it should be "0 1 6 1 0 2 7 2 0 3 8 3"
- Then list the blocks in the right wheel in clockwise direction, again from the block next

to the shared blocks, but without listing the shared blocks again. E.g., for the natural configuration, it should be "0 4 9 4 0 5 10 5 0".

Each of your programs should report the shortest path to reach the natural configuration, which should contain lines describing the move (left or right, clockwise or counterclockwise). E.g., given the configuration on the right above (i.e., "8 3 0 1 6 1 0 2 7 2 0 3 0 4 9 4 0 5 10 5 0"), your program should output a line `left clockwise`. In your answer, report the amount of time that your program uses to solve a puzzle that requires 1 to 24 steps (or that the puzzle cannot be solved if it uses longer than 10 seconds or more than 40MB memory).

Note: you can easily use up the memory of the computer with BFS or BDS. You might want to limit your memory using using the `ulimit` command (e.g., `ulimit -v 40960`), so that using excessive memory will crash your program rather than hang your computer.

3. **A* search (20%)** The travelling salesman problem (TSP) is a problem to find a path that passes through each nodes in a graph exactly once. In this question we try to optimally solve the geometric TSP problem, where the nodes in the graph are points in the plane, and every pair of points are connected by an edge with the geometric distance as the edge cost. To simplify the work, we do not require the path to start and end at the same node, and we fix the first node visited by the travelling salesman. The problem is known to be NP-hard, so there is no known polynomial time algorithm for the problem.
 - a. Given a graph of nodes and weighted edges, its Minimum Spanning Tree (MST) is a tree composed only of edges in the graph which has the minimum total cost. Use it to make a admissible and consistent heuristic for TSP, and prove that it is admissible and consistent.
 - b. In the web page you can find a program which solves the problem using Uniform Cost search, and another program which calculate the MST of a set of nodes using n^2 time for n nodes. Use them to implement A* search as you suggest in part (a). A program is also provided for generating a random set of points. Compare the speed of the original program against your program on generated sets of n random points, where n is 5, 10, 15, 20, 25 and 30 (reports that the problem cannot be solved if it requires more than 10 seconds or 40MB memory). Run your program multiple times using different random input in order to obtain a more accurate result.
4. **Local search (35%)** It is possible to solve the problem in question 3 suboptimally by the idea of local search: a path is represented as a permutation of the node number, and in each step we modify the path slightly (e.g., swap two points).
 - a. Implement hill climbing with random restart, dealing with plateaus in the following way: when the maximum improvement is less than 1%, randomly walking on any path with an improvement or a degradation of at most 1%. Try to improve the running speed of the program as far as you can. Compare your answer with the result in question 3 to determine how good is the answer produced, when the amount of time used is no more than 1 second.
 - b. What is the major obstacle in making your program run even faster?