

Written Assignment #1: Sample Solutions

Question 1 (3 points). PEAS descriptions.

- a) A web-based machine translation system:
Performance: Speed of computation; quality of translation (e.g. approximated by BLEU score).
Environment: Web server, handling text in multiple languages; Users.
Actuators: Text output (e.g. via HTTP)
Sensors: Text input (e.g. via CGI)
- b) A search-and-rescue bot for exploring a collapsed mine:
Performance: Effectiveness in locating survivors; maybe assisting.
Environment: Rubble, dirt, dust, machinery, miners.
Actuators: Wheels, claws, lanterns, wireless transmitter, etc...
Sensors: Cameras, GPS, wireless receiver, laser, range-finders, air-quality analyzer, etc...
- c) A burrito-building robot:
Performance: Satisfy customer; minimize food waste; speed.
Environment: Food ingredients, kitchen equipment.
Actuators: Conveyor belts, jointed arms and hands.
Sensors: Temperature probe, timer; ingredient quantities, positions.

Question 2 (3 points). Graph Search

- a. A B D C E B D F (when nodes are placed on stack in reverse alphabetical order and thus popped off in alphabetical order, it is a trivial search if you place nodes on stack in alphabetical order)
- b. A B C E F (assuming nodes are placed in queue in alphabetical order)
- c. A B C D E F (closed list disambiguates the very last priority queue operation in this example).

Question 3 (4 points). Burrito assembly scheduling

- a. There can be multiple employees, but they should not be viewed as autonomous agents because they are neither competing nor cooperating in order to maximize their individual performance measures; rather, they are working towards a single shared performance measure. The employees are simply resources, and in this scheduling problem the single planner is the manager who applies these resources.
- b. Problem formulation: Here is one way to provide an incremental formulation, where states represent partial schedules that can be augmented. Here the successor function is particularly unspecific about what task/time pairs to add, and could represent an infinite branching factor.
 - States: a state is represented by a schedule, a set of task/time pairs
 - Initial State: {}
 - Successor Function: Generate new states by adding a task/time pair to the current schedule.
 - Goal Test: Check for the following:
 - Are all tasks on the schedule?
 - For each task, do all of its prerequisite tasks complete before it is started?
 - Are there no more than e tasks being worked on at any time instant?
 - Cost Function: The cost of a schedule is the time at which the last task will complete.
- c. Heuristics are often derived by creating a relaxed problem :
 - Suppose we relax the constraint that an employee can only perform one task at a time. Given a current schedule, assign all remaining unscheduled tasks to the employee with the earliest availability. Consider the longest chain of unscheduled tasks such that each member requires the previous task; when this chain is completed, all tasks will have finished. This gives the optimal solution cost for the relaxed problem, which is an admissible heuristic for the original problem.
 - Additionally, we could relax the constraint that a task can only be started after its prerequisite is completed. Then the heuristic would be determined by the largest cost of any remaining unscheduled task. This is a more relaxed problem, and a less useful heuristic.
- d. For this problem formulation, only the state description and successor function would change:
 - States : a state is represented by a schedule , a set of task/time/employee triples
 - Successor Function : Generate new schedules by adding task/time/employee triples

Question 4 (5 points). The *Traveling Salesperson Problem* (TSP)

- a. Without loss of generality, let s be the canonical start/end city of all tours. There are $N - 1$ choices for the second city, $N - 2$ for the third, etc... so $(N - 1)!$ paths that start and end at s , visiting each city in C exactly once. A tour and its reversal are equivalent, so there are $(N - 1)! / 2$ distinct tours.
- b. Problem formulation:
- States : Let a state be represented by a path p , a list of cities.
 - Initial State : (s) , for some $s \in C$.
 - Successor Function : Generate new paths by appending c to the current path, for each $c \in C$.
 - Goal Test : Check if the path is a valid tour.
 - Cost Function : The cost of a path is $\sum d(c_i, c_j)$, for all c_i followed by c_j in the path.
- c. For this formulation, DFS is not complete – even with a graph-search that checks for repeated states. (The incremental successor function generates states at depth $d + 1$ which are necessarily different from any states at depth d or above and which might be on the closed list.) So DFS could (and almost certainly will) proceed down an infinitely long branch of the search tree. Given the finite branching factor N , BFS is a complete search algorithm, and optimal since the step costs are all equal. Note that all solutions are found at a depth of N in the search tree. Exploiting this knowledge of the problem, a better complete algorithm to employ would be a depth-limited variant of DFS.
- Alternative answer:** The successor function could choose $c \in C \setminus S$, where S are the cities that already appear on the current path; i.e. no cities will repeat on any path. This search tree is finite (with maximum depth N) so DFS will be complete – and optimal since all solutions cost the same. DFS would be superior to BFS in this case because it would expand fewer nodes.
- d. If step costs are strictly positive (consider: $d(c, c) \neq 0$), then uniform-cost search will be complete and optimal. In addition, A* search will be optimal if we use an admissible heuristic.
- e. A very basic heuristic estimates the cost to the goal as $h = d(e, n)$, where e is the last city in the current path and n is its nearest-neighbor. From the current state, the optimal solution must begin with a first step of distance no less than $d(e, n)$; thus the heuristic is admissible.

A more complicated heuristic... Let S be the cities that appear on the current path. The heuristic is the weight of the minimum spanning tree (MST) for the subgraph that includes cities in $(C \setminus S) \cup \{s, e\}$. From the current state, the optimal solution will connect all the vertices of this subgraph, and that path is a spanning tree. The MST cannot cost more than any other spanning tree, so this heuristic is admissible. The MST can be efficiently computed with Prim's algorithm or Kruskal's algorithm.