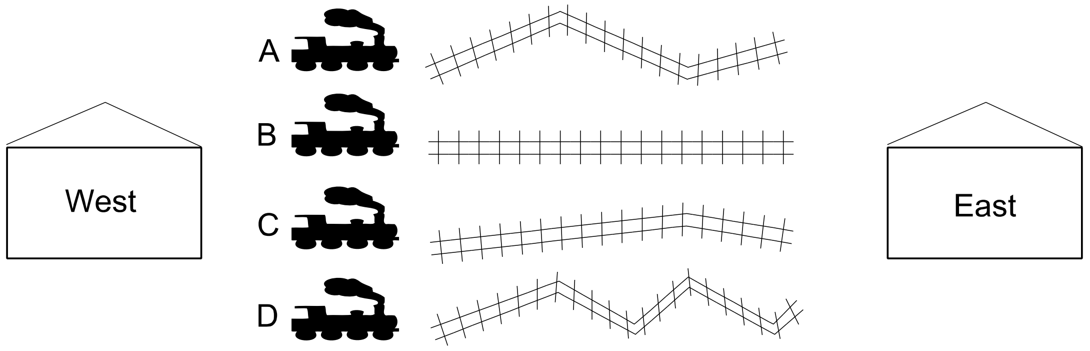
EECS 492 – Winter 2014
Problem Set 2
Due: Monday February 10, midnight.

Overview

This assignment covers topics in problem-solving as search, including uninformed and informed search methods, local search methods, and constraint satisfaction problems. It will ask you to formalize problems so that you may apply search methods to solve them, and to apply those methods. Some problems will require you to apply these methods by hand while others will require you to implement them as a computer program.

Task 1: Search Formulation (20 pts)

You and a co-worker need to move trains from the West train depot to the East train depot. There are four trains at the West depot that need to be moved to the East and each is on different track. The trains cannot be moved to a different track. Each track is a different length and therefore the amount of time it takes to travel varies between each train:



Train	A	B	C	D
Time (minutes)	40	15	20	60

To transport a train, a person needs to climb inside it and drive to the destination. You and your co-worker may take separate trains or ride in the same train. Before leaving on the next trip, you need to wait for both people to be at the same depot. In other words, you will have to wait for the slower person and then both people have to start on the next trip at the same time. Assume no time is spent at either depot and you are able to leave immediately when both people get to a depot. You must get all four trains to the East.

Formulate this as a search problem by providing a detailed description of the following:

a. A specific state representation that captures important details and leaves out unimportant ones. That is, you should be able to *uniquely* represent every meaningful state of this problem. Give an example of how you would represent the state where trains A and B are at the West depot and trains C and D and the people are at the East depot.

b. The initial state, expressed in your representation.

c. The goal test, expressed in terms of your representation.

d. A description of the possible actions, including in which states they are allowed and what their effects are. Your description should be in terms of your representation. Explicitly state how many possible actions there are.

e. A description of how to compute the path cost for any sequence of actions. Assume this only involves travel time.

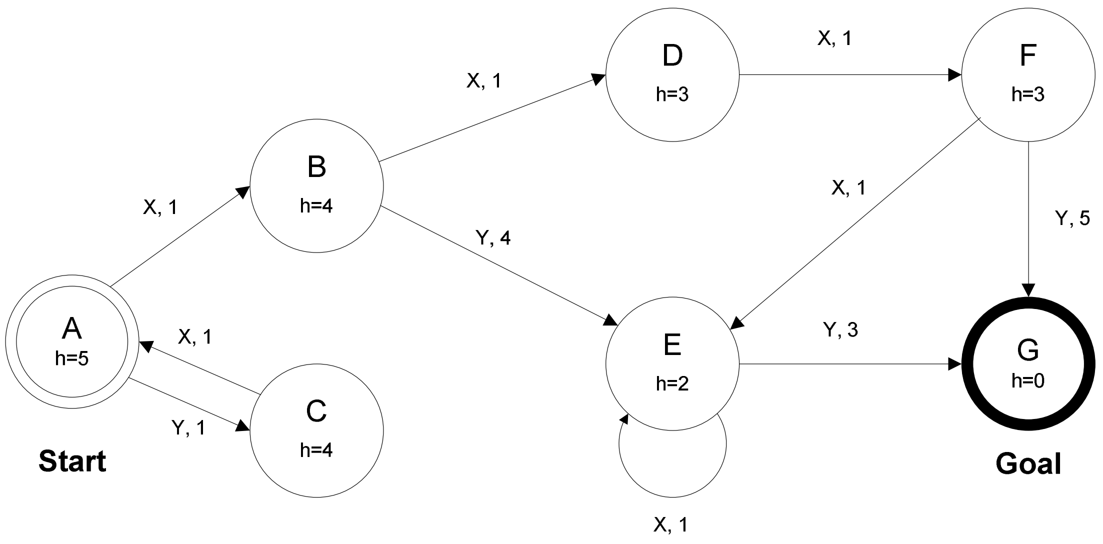
f. Find an action sequence that transports the trains in the least amount of time. It is not necessary to compute this by examining all possible action sequences. You do not need to show exactly how you got this answer, but describe why it makes sense or your reasoning behind this conclusion. How long does this solution take? (Hint: Note that more time spent waiting for one person to catch up with the other is probably not time well spent)

g. Give a possible heuristic that we might use for an informed search algorithm. The heuristic should be admissible and non-trivial (e.g. don't use h(n)=0 for all n).

Task 2: Uninformed and Informed Search (30 pts)

The components of a search problem can be encoded into a state space graph like the one shown below.

- Each vertex (A-G) represents a state of the world.
- The initial state is labeled with "Start".
- All states that satisfy the goal test are labeled with "Goal".
- Directed edges show which actions (X or Y) transition between which states. The cost of each action is also shown.
- Values of a given heuristic are shown (e.g. h=5) for each state.



These problems will deal with executing various algorithms to get to the goal state from the start state. Use the general search process discussed in class (goal test only applied on examination/expansion), summarized in the textbook in Figure 3.7.

When writing out your solutions to the problems below, use this notation:

- You can indicate a node by the path it represents (e.g. ABD)
- Write out the fringe like this: [AB, AC]
- The "front" of the fringe is on the left. In the above, AB would be expanded next.

When expanding a node and generating its successors, assume the available actions (X or Y) are processed in alphabetical order. Here are two examples:

Expanding Node ABE with Breadth First Tree Search

Queue	Nodes Examined/Expanded	Successor Nodes
[ABE, …]	ABE	ABEE, ABEG
[…, ABEE, ABEG]		

Expanding Node ABE with Depth First Tree Search

Queue	Nodes Examined/Expanded	Successor Nodes
[ABE, …]	ABE	ABEE, ABEG
[ABEE, ABEG, …]		

To show your work, show which node is expanded and which successor nodes are generated at each step. Also write out the contents of the fringe after and, if appropriate, the contents of the explored set. If a successor is generated but immediately discarded or if a node on the fringe is replaced, cross it out. As an example of the format we're looking for, here's the execution of a breadth first graph search:

Breadth First Graph Search

Queue	Nodes Examined/Expanded	Successors	Explored Set
[A]	A	AB, AC	{A}
[AB, AC]	AB	ABD, ABE	{A, B}
[AC, ABD, ABE]	AC	~~ACA~~	{A, B, C}
[ABD, ABE]	ABD	ABDF	{A, B, C, D}
[ABE, ABDF]	ABE	~~ABEE~~, ABEG	{A, B, C, D, E}
[ABDF, ABEG]	ABDF	~~ABDFE~~, ABDFG	{A, B, C, D, E, F}
[ABEG, ABDFG]	ABEG		
Upon examining ABEG, we see that it passes the goal test and so return **ABEG** as the solution.			

If something goes wrong that prevents your algorithm from finding a solution, show your work up to that point and include a description of why the algorithm fails.

Show the execution of the following search algorithms:

a. Breadth First Tree Search
b. Depth First Tree Search
c. Iterative Deepening Tree Search (show all iterations)

Now, we will consider priority-first (textbook calls these "best-first") algorithms that use an *evaluation function* f(x) to prioritize node examination/expansion. If node ACF has $f(ABE) = 5$, you should now write it as ABE(5). When adding new nodes to the fringe, first add them to end and then make sure to sort the nodes in a *stable* fashion. That is, if two nodes have equal priority, the one that has been in the fringe longer will be examined/expanded first. And if two successor nodes have the same value, they will appear in their generated order (as in the examples above).

Show the execution of the following search algorithms:

d. Uniform-Cost Graph Search
e. Greedy Priority-First Tree Search
f. A* Graph Search

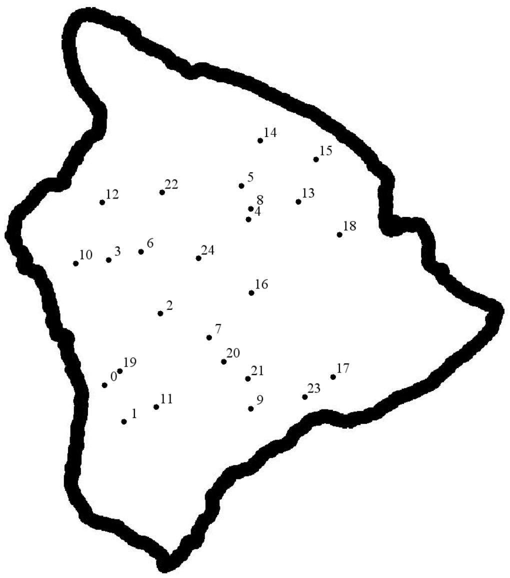
The heuristic in the graph above turns out to be admissible (and consistent). Consider specifically the heuristic value for state F, h(F)=3. Find a different (nonnegative real) value for h(F) that would make the heuristic:

g. Inadmissible
h. Admissible but inconsistent. And show why it is inconsistent.

Task 3: Local Search (30 pts)

For this task, you will implement a local search method to find approximate solutions to an instance of the Traveling Salesperson Problem (TSP). (http://en.wikipedia.org/wiki/Travelling_salesman_problem) You'll also investigate how different successor functions affect search efficiency and solution quality.

Let's say you need to visit each of the 25 cities shown on the map below and would like to find the most efficient route. You may start in any location but must visit each other city exactly once and return to finish in the same city as you started. This is called a "tour" of the cities.



You should use a complete-state formulation where the state space is the set of possible orderings of the integers 0-24. Technically some orders will map to the same tour (e.g. 0,1,2,3 and 1,2,3,0 and 3,2,1,0), but this won't be an issue. Your goal is to find a minimal tour, so your objective function will naturally be the total travel time of the tour (minimize it). For example, the cost of the tour 0,1,2,3 is the time to get from 0 to 1, then from 1 to 2, then from 2 to 3, and finally from 3 back to 0. The travel times between each pair of cities are given as a (symmetric) matrix of comma delimited values in the `cities.csv` file. The j^{th} element of the i^{th} row is the cost to travel between cities i and j.

For this task, you'll consider two different successor functions:

- City Swap
  The successors of a tour (state) are all tours that result from switching the position of two cities in the ordering. For example, the 4 city tour 1,2,3,4 could be changed into 2,1,3,4 or 3,2,1,4 or 4,2,3,1 or 1,3,2,4 or 1,4,3,2 or 1,2,4,3.

- 2-Opt
  The successors of a tour (state) are all tours that result from reversing any substring of length L > 1. For example, the 4 city tour 1,2,3,4 could be changed into 2,1,3,4 or 1,3,2,4 or 1,2,4,3 or 3,2,1,4 or 1,4,3,2 or 4,3,2,1.

Given the above, you will implement a local search algorithm in a language of your choosing. You should include the data and/or answers required for the problems below with the rest of your assignment, but please also turn in a copy of your source files. Code that is well organized and documented greatly increases your chance for partial credit! You may use any programming language you like (though if it is too exotic you should tell us how we can run your program ourselves if we choose to do so); please include a comment specifying the language at the beginning of each file. You are also welcome to use your language's "standard library" to the extent the functionality it provides does not directly pertain to local search or the TSP.

a. Write code to implement the objective function using the `cities.csv` file and use it to calculate the cost for the tour 0,1,…,24.

b. Write code to generate random tours (orderings) and combine with the code from part a. to find the costs of 10000 random tours.
   i. What is the mean cost?
   ii. What is the lowest cost found?

c. Implement the City Swap successor function and use it to determine which successor of the tour 0,1,…,24 has the lowest cost. Write down the successor's ordering and cost.

d. Repeat part b. using the 2-Opt successor function.

For the next parts, you will be implementing steepest-descent local search (the minimization equivalent of hill-climbing search). You may break ties between successor tours with equal costs arbitrarily – it will not affect your results. (Suggestion: Read through all the subtasks below before starting so you can formulate an implementation that can accommodate the various requirements.)

e. Implement steepest-descent local search using City Swap as the successor function and the in-order tour (0,1,…24) as the initial tour. Run the algorithm for 50 iterations (or until convergence). Write the resulting tour and its cost.

f. Now introduce stochasticity into your steepest-descent local search implementation. Introduce into the algorithm in part e. the capability of random (re)starts, where it starts with a randomized initial tour rather than the fixed initial tour. Run the algorithm 100 times with the numbers of random (re)starts given in i and ii below. Record the number of runs out of the 100 that have a tour cost below 320. (Note: You will also have to compute average cost in part h.)
   i. 1 random start (i.e. 1 full steepest-descent local search from a randomized start tour)
   ii. 10 random (re)starts (i.e. 10 full steepest-descent local searches each with a randomized start)

g. What do you notice about the effect of having more random restarts? What does this tell us about our starting location for the search?

h. Using City Swap steepest-descent local search with 10 random (re)starts, find the average resulting tour cost over 100 runs.

i. Repeat part h. using the 2-Opt successor function.

j. Given your answers to parts h. and i., which successor function worked the best? The 2-Opt method is well known for local search in the TSP. Search online to find out a little more about it and briefly describe (in your own words) why it works well for finding improved tours.

Task 4: Constraint satisfaction (20 pts)

You are scheduling a concert that lasts eight hours. There are four bands that will be playing at the concert and you need to figure out how to plan the concert in order to meet their available times as well as each band's various stage equipment needs.

Each band will play two sets of songs, each lasting exactly one hour. These are the times they are available to perform as well as their equipment needs:

Band	Hours Available	Extra Microphones	Laser Lights
A	1, 2, 3, 5, 8		✓
B	3, 4, 6, 7		✓
C	1, 2, 3, 7, 8	✓	
D	1, 2, 4, 5, 6	✓	

Additionally, because you are sharing some stage equipment with other venues, there are availability constraints on each resource. If a band needs one of these resources to perform, then they can only perform when these are available. These are the only times you are able to provide these resources:

Resource	Hours Available
Extra Microphones	1, 2, 3, 4, 5, 8
Laser Lights	3, 4, 5, 6, 7, 8

Formulated as a CSP, this problem would have 8 variables, one for each timeslot (call them 1, 2, 3, 4, 5, 6, 7, 8). Then a partial assignment would look something like

(B,A,-,-,-,A,-,-) or (1=B, 2=A, 6=A)

which would mean that band A performs during hours 2 and 6 and band B performs during hour 1. The other hours have not yet been assigned.

a. Show the initial domain of a variable (each variable has the same initial domain). The initial domain is the domain prior to applying any constraints.

b. Apply the bands' time availability constraints and show the resulting domains for each variable. Why is this not yet node consistent?

c. Finish applying node consistency and show the resulting domains.

For the next parts, you will be tracing the steps for solving the CSP. Show each assignment and backtrack step individually. Assign variables in numerical order except when MRV is being used (MRV order takes precedence). You should assign values to variables in alphabetical order. For example, if you had the following CSP:

Variables	Values
1	{A, B}
2	{A, B}

Your answer would look like

Queue	Assignment
1, 2	(A,-)
2	(A,B)

d. Starting with the domains found in part c., perform a backtracking search without forward checking to find a satisfying assignment. Show the queue just before each step of the search and the assignment after that step is taken, and clearly indicate where backtracking occurs.

e. Do the same thing in part d. but use a backtracking search using the MRV heuristic with forward checking to order the variables. Ties in the MRV heuristic should be broken so that the variables are assigned in numerical order. Show the queue at every step and the assignment after that step is taken, and clearly indicate where forward checking changes a domain and where backtracking occurs.