Intelligent Agents

Practical 5 – Local Search

PART 1 – Search

These questions are designed to ensure that you understand the core concepts of informed search. Please attempt the questions before the practical so that they can be discussed there.

1. One of the big differences between graph search and local search is that the former (as the name implies) searches through a graph that represents some aspects of the search space while local search tends to search through a search space directly constructed from the world state. Consider how this applies to the instance of Vacuum World discussed in practical 1.
    a. Decide on a way to represent the world state and what the possible transitions are.
    b. Decide on a "goodness value" (which may be the same as the performance measure). Are you maximizing or minimizing?
    c. Use Hill Climbing on this problem.
    d. Is local search appropriate for this problem. Why?

2. Apply local search to the 8-puzzle.
    a. Decide on a way to represent the world state and what the possible transitions are.
    b. Decide on a "goodness value" (which may be the same as the performance measure). Are you maximizing or minimizing?
    c. Use Hill Climbing on this problem.
    d. Is local search appropriate for this problem. Why? Can you generalize to determine what sort of problems Hill Climbing may be more or less appropriate for?

3. Consider the task of scheduling classes in our labs. This is a subset of the University Timetabling Problem. You may want to look at a much smaller instance (say 3 rooms, a smaller number of "slots" and simpler units with attached staff members). Assume that a scheduled class that is not in a hard clash (there isn't another class scheduled in the same room at the same time and the staff member assigned isn't scheduled in another slot at the same time) gets 5 points, which rises to 7 points if there isn't another class for the same unit scheduled at the same time.
    a. Decide on a way to represent the world state and what the possible transitions are.
    b. Look at using Hill Climbing to solve this problem. Depending on how the classes and staff line up there may or may not be local maxima.
    c. Look at using a Genetic Algorithm or Simulated Annealing to tackle this. How would you set that up? (You don't need to do this fully but should at least look at the preparation.)

PART 2 – Coding & Preparation

Part of Local Search is setting up the problem, and different problems are going to have very different ways of approaching them. For example, consider 8 Queens:

- You need to decide how to record the problem state. In this case that's related to a Chess board and the placement of the queens.
- You need to decide what the valid transitions are. In this case that will be moving one

queen.

- You need to be able to calculate the "goodness" of each state (and do so efficiently). Often, we calculate the metric for the initial state and then only modify it based on the transition, to avoid re-calculating it. In this case, when moving a queen, we would remove any points from collisions at the old location and add in points for collisions at the new location.
- Local search functions only generate the "neighbouring" states for each step and don't keep them stored. That means you'll need an efficient way of storing the states for each step – this is where you'll normally use a priority queue.
- The search itself is a way to take the current state (or states – remember to allow for a population of "current states") and generate a new population of states (which may involve storing them on the priority queue and only taking one to the next step). Hill Climbing always takes the best choice (the first item in the priority queue) but other methods may randomly choose an item from the queue, possibly based on priority.

You will note that it's not feasible to load information from a file this time because the implementation will be fairly specific. However, you do want to maximize code re-use. In this case you can use good OO programming practice along with polymorphism to create a general and specific class for most of the above. Doing this allows you to fully re-use some code (like your priority queue).

For example, you can have a general State class and then a specific QueensState class that specializes State. State would have a goodness measure but the way to calculate it would be a stub that is completed in QueensState. That means that the priority queue could act solely on State (since it needs only the number, not how that number came about).

Code up the 8 Queens problem with Random Restart Hill Climbing to maximize this code re-use. You will not be doing 8 Queens for the assignment but have a variant in the test (you can extend this as part of your test preparation if you wish) and you can re-use the general part of the code for the assignment.