Simulated Annealing

The travelling salesperson problem (TSP) is a difficult integer program to solve. The solution that we discussed in class is actually not a very efficient way to solve it when the number of nodes increases. When the number of nodes is quite large it is almost impossible to find the truly optimal path. One approximation methodology to solve this problem is called simulated annealing.

Simulated Annealing

Simulated annealing works by first initializing a randomly chosen path. Then we go through and repeatedly look at possibly changing portions of the path. There are two types of changes considered: reverse and transport. For a reverse change, you randomly pick 2 nodes, 1 is labeled the starting node, and 1 is labeled the ending node; let's say these nodes are i and j. We find the segment of the path that starts at node i and goes to node j, disconnect it from the entire path, reverse it and put it back into the path. For example, if we had a 6-node problem and we have a path that goes 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 0, and randomly picked the starting node to be node 5 and the ending node to be 2, then the segment of the path that connects 5 to 2 is 5 -> 0 -> 1 -> 2. To reverse this path, we find the node that used to go into 5, and make it go into 2, and find the node that used to come after 2 and make it come after 5, so now the new path after reversing would 4 -> 2 -> 1 -> 0 -> 5 -> 3 -> 4. The other type of change to the path, transport, similarly finds a random start and end node and finds the path between those 2 nodes. Then a transport change removes that sub-path from the loop, closes the loop with the remaining nodes, finds a random link on new loop, splits the loop there and pastes the sub-path. Going back our earlier example, we would remove the 5 -> 0 -> 1 -> 2 sub-path, and close the loop on the remaining nodes, to get 3 -> 4 -> 3. We then choose one of these links at random to cut, let's say 3 -> 4, and put the sub-path in where that link was, to get 3 -> 5 -> 0 -> 1 -> 2 -> 4 -> 3.

With these two types of changes, we can now put them together to describe the simulated annealing policy. 1) Initialize a loop completely at random. 2) Toss a fair coin and pick randomly between a reverse and transport change. 3) Try the change. 4) If it reduces the total distance travelled keep the change and go back to 2; if the change does not decrease the distance travelled don't keep it and go back to 2. As described, this is a greedy algorithm and may get stuck in a local minimum. An easy way to fix this issue is to randomly accept changes sometimes even if they increase the distance. When a change increases the distance, instead of not accepting it we can toss a weighted coin (more on the weight to come), and if it comes up heads keep the change even though it leads to an increase in distance travelled. We don't want the probability of accepting very bad changes to be very high. We also want the probability of accepting bad changes to get smaller as the algorithm progresses. To accomplish these goals, pick some initial value of $\epsilon \leq 1$. For each change that increases the distance call that increase in path distances d, then set the probability of accepting a bad change equal to $\epsilon/\exp(d)$. Then after each iteration of the algorithm, shrink ϵ a little bit, perhaps by multiplying it by 0.99. Then after many iterations of this algorithm, you should have a pretty good path. This is a heuristic algorithm and is NOT guaranteed to converge to the optimal path, but typically does a pretty good job.