Function Optimization With Local Search– Solution

1) In the N-Queens problem, we want to place N queens on an $N \times N$ board with no two queens on the same row, column, or diagonal. Come up with a value function and use hill climbing to try to solve the problem by minimizing this value function, starting with the configuration given below. Generate the successors of a state by moving a single queen vertically.

	A	B	C	D
1				
2	Q1		Q3	
3		Q2		Q4
4				

Answer:
The value function is the number of pairs of queens that can attack each other, assuming no other queens are on the board (that is, three queens in the same row count as three pairs, not two). In the initial state, we have the pairs (Q1, Q2), (Q1, Q3), (Q2, Q3), (Q2, Q4), (Q3, Q4), making its value 5.

Moving Q1 to A1 removes the pairs (Q1, Q2) and (Q1, Q3), decreasing the value by 2. Moving it to A3 removes the pair (Q1, Q3), but adds the pair (Q1, Q4). Moving it to A4 does not change the pairs. A symmetric argument can be made for Q4.

Moving Q2 to B1 does not change the pairs. Moving it to B2 removes the pair (Q2, Q4). Moving it to B4 removes all three pairs that Q2 appears in. A symmetric argument can be made for Q3.

The lowest value we can get with one move is 2, either by moving Q2 to B4 or Q3 to C1. Moving Q2 to B4, we get:

	A	B	C	D
1				
2	Q1		Q3	
3				Q4
4		Q2		

There are many possible moves in this state, but moving Q3 to C1 finds the solution (that is, a state with the lowest possible value of 0), so we can skip the other moves:

	A	B	C	D
1			Q3	
2	Q1			
3				Q4
4		Q2		

2) How would you approach the Traveling Salesman Problem if we wanted to find a good (but not necessarily the best) solution to it using hill climbing?

Answer:
We start with a random solution that visits all cities. The value function is the total distance traveled by the salesperson. We can generate the successors of a state by picking any two cities on the tour and exchanging them. The 2-opt and 3-opt algorithms are more complex but very well performing hill-climbing algorithms for the Traveling Salesperson Problem. The 2-opt algorithm was discussed in class and picks any subpath of the tour and reverses the order of the cities visited on this subpath.

3) What are the advantages/disadvantages of local search methods (such as hill climbing and simulated annealing) compared to A*? For which kind of optimization problems should local search be preferred?

Answer:
Local search methods are typically much faster than A*, although they do not provide any optimality guarantees. They also use little memory, since they only need store a single state. Local search methods are best applied to computationally hard (e.g. NP-hard) optimization problems. One thing to note is that local search algorithms search in the space of candidate solutions. For instance, if we apply local search to find short paths on graphs, we need to generate an initial path (possibly by searching the graph with depth-first search or even suboptimal versions of A* search), and then start optimizing this path. In this scenario, local search is a poor substitute for A* search.

4) We want to find a local minima of the function $f(x) = 2x^3 - 3x^2 - x + 1$. Start with $x = 0$ and apply three iterations of the gradient descent algorithm, using a learning rate of $\alpha = 0.2$.

Answer:
Remember the update rule for the gradient descent algorithm:

$$x := x - \alpha df(x)/dx$$

Taking the derivative of $f(x)$ with respect to x, we get:

$$df(x)/dx = 6x^2 - 6x - 1$$

Plugging in α and $df(x)/dx$, our update rule becomes:

$$x := x - 0.2(6x^2 - 6x - 1)$$

$$x := -1.2x^2 + 2.2x + 0.2$$

With this update rule, the first 10 iterations of gradient descent is as follows:

Iteration	x	f(x)
0	0.000000	1.000000
1	0.200000	0.696000
2	0.592000	-0.228443
3	1.081843	-1.060652
4	1.175593	-1.072266
5	1.127882	-1.074638
6	1.154799	-1.075492
7	1.140285	-1.075724
8	1.148327	-1.075798
9	1.143933	-1.075819
10	1.146353	-1.075826

5) What are the advantages and disadvantages of carrying over the fittest two individuals to the next generation?

Answer:
Carrying over the fittest individuals (also known as 'elitism') tilts the genetic algorithm towards local search, because the best individuals stay around to (mostly) create more and more children very near to themselves in the genotype space. This could make the genetic algorithm more efficient, but also can cause it to more easily get caught in local optima.

6) What are the advantages and disadvantages of running genetic algorithms with only mutations and no crossovers? How about only crossovers and no mutations?

Answer:
Genetic algorithms without mutations cannot explore outside the possible points in space formed by choosing any combination of parameter settings in its original population. As individuals are weeded out, this combination reduces, ultimately to a single individual. For instance, if the individuals are represented as bit strings and the first bit of every individual is 0 in the first generation or becomes 0 (if all the 1's are eliminated), then the algorithm cannot generate solutions whose first bits are 1.

Genetic algorithms without crossovers perform purely random searches whose children are very similar to their parents. Crossovers help rapidly transfer high-performing parameters throughout the population, making genetic algorithms more efficient in problems where the parameters are at least somewhat independent of one another.

So, in general, mutations keep the search (somewhat) global and crossovers make it more efficient.

7) How would you encode a state if you were using a genetic algorithm to solve the Traveling Salesman Problem but only wanted to use a straightforward crossover operation that switches prefixes of both parents' encodings (that is, we randomly pick a cutoff point in the encoding, use the encoding of the first parent up to that cutoff point, and use the encoding of the second parent after that cutoff point)?

Answer:
We can, for example, assign an integer to each city, encoding its priority. These priorities correspond to a unique tour, where the salesman visits the cities in the order of their priorities (breaking ties lexicographically). For this encoding, crossovers and mutations always create tours and thus valid solutions. Compare this encoding to one where, for each vertex, we keep the next vertex to visit. The crossover operator would have to be defined very carefully in order to generate valid solutions.