Example – TSP with EC methods

The Traveling Salesman Problem (TSP) is a classic optimization problem in computer science and mathematics (game link). It can be described as follows. Given a list of cities and the distances between each pair of them, the goal is to find the shortest possible route that:

1. Starts at a city A (or any other starting city),
2. Visits every other city exactly once, and
3. Returns to the city A (or other starting city).

TSP is an NP-hard, meaning no known polynomial-time algorithm can solve it exactly for large numbers of cities. However, there are approaches, like Concorde (link), that are extremely good even for large instances. In a symmetric TSP, the distance from city A to B equals the distance from B to A. The number of possible tours grows factorially - for n cities, there are $(n-1)!/2$ possible routes (for symmetric TSP).

One possible approach for TSP is to use genetic algorithms. Here, usually, the underlying data structure is a permutation vector of length n (or $n-1$ if we decide to have a designated starting city). Normal mutation and crossover operations will not work: 'bit flips' do not make sense, 'stitching together' to permutation vectors 'as is' (i.e., normal crossover) will not produce a permutation vector.

For mutation, a simple operation that can be used for a mutation is a swap. On a permutation vector of length 5, it would look like this

before mutation	swap position 2 and 4	after mutation
$A \to E \to B \to C \to D$		$A \to C \to B \to E \to D$

For crossover, there are also a few possibilities (wiki link). One of them is the Partially mapped crossover (the second child can be constructed by swapping P_0 and P_1):

Procedure	**Example**	**Example Chromosome**
	Let be given two permutations of the same set.	$P_0 = (A, B, C, D, E, F, G, H)$ and $P_1 = (C, G, E, A, F, H, B, D)$
Randomly select two crossover points forming a gene segment in P_0	Here from gene position 4 to 6.	$P_0 = (A, B, C, \underline{D, E, F}, G, H)$
The selected section is copied to the child chromosome in the same position.	The open positions are indicated by question marks.	$P_C = (?, ?, ?, \underline{D, E, F}, ?, ?)$
Look for genes that have not been copied in the corresponding segment of P_1 starting at the first crossover point. For each gene found (called m, look up in the offspring which element (called n was copied in its place from P_0. Copy m to the position held by n in P_1 if it is not occupied. Otherwise, continue with the next step.	Gene A is the first uncopied gene in the corresponding segment of P_1 : $(..., A, F, H, ...)$. Gene D was copied from P_0 in its place in P_C. The position of D in P_1 is the furthest right position and A will be placed there in P_C.	$P_C = (?, ?, ?, \underline{\mathbf{D}, E, F}, ?, ?)$ $P_C = (?, ?, ?, \underline{D, E, F}, ?, A)$
If the place taken by n in P_1 is already occupied by an element k in the descendant, m is put in the place taken by k in P_1.	The next gene in $(..., A, F, H, ...)$ is F and this has already been copied into the child chromosome. Thus, the next gene to be handled is H. Its position in the offspring would be the position of F in P_1. However, this place is already occupied by gene E. So H is copied to the position of E in P_1.	$P_C = (?, ?, H, \underline{D, E, F}, ?, A)$
After processing the genes from the selected segment in P_1, the remaining positions in the offspring are filled with the genes from P_1 that have not yet been copied, in the order of their appearance. This results in the finished child genome.	The genes copied from P_1 are C, G and B.	$P_C = (C, G, H, \underline{D, E, F}, B, A)$

A different set of approaches are the component-based ones, that try to construct good solutions (permutation vector/tours) from a set of possible components (like GRASP or Ant colony optimization). Here, the components are simply the possible edges between all the cities (and their cost is the distance between those cities).

One possible hill-climbing strategy to improve on the constructed (or "breeded") solutions is to use the so-called 2-opt (link) heuristic. The idea (called a 2-opt move) is to start with any tour and then, repeatedly:

1. Pick two edges in the tour.
2. Remove them.
3. Reconnect the two resulting paths in the opposite order, which reverses a segment of the tour.
4. Accept the change if it shortens the overall tour.

The 2-opt move is repeated until no improving 2-opt move exists (with a worst-case runtime of $O(n^2)$ checks per iteration, a full run may take approximately $O(n^3)$ time.) On a permutation vector a 2-opt move selects two indices i and j in the permutation:

$$1 \leq i < j \leq n$$

Then it reverses the segment of the permutation between them, producing a new tour.

before 2-opt move	position 2 and 5	after 2-opt move
$A \rightarrow E \rightarrow B \rightarrow C \rightarrow D \rightarrow F$		$A \rightarrow D \rightarrow C \rightarrow B \rightarrow E \rightarrow F$

In the file `ex12.mat` you will find an instance of symmetric TSP with 30 cities (positions of cities on a map and a distance matrix).

Assignment: Implement two heuristics, GA and GRASP with the mutation/crossover/hill-climber as described above. With a limited number of possible tour evaluations (max 1e5), what was the setting of the algorithms (population size, tournament size, percentage of components in iterations, number of hill-climbing moves) produced best results?