meaning that if the history happened to be $(CD)(DC)(CC)$ this player would cooperate on the next turn. If the history was $(DD)(DC)(CD)$ the player would also cooperate on the next turn, but if the history was $(CD)(CD)(CD)$ the player would defect on the next turn.

Now we can find a play for each different history but what happens at the start of the game when there is no history? Each different player (or chromosome) is given an hypothetical history so that it can generate its first play. That is, we add six more genes to the start of the chromosome to act as the 'last' three plays. The player then uses the play that has been assigned to that particular history. So we end up with a string of 70 genes as our chromosome.

The fitness of each chromosome is found by playing against other players. The usual one or two point crossover works for these chromosomes and mutation is also just the usual mutation routine. Notice that the chromosomes are simply binary strings but we have the letters C and D rather than the binary digits 0 and 1.

2.4 Encoding

In this section we will investigate possible ways to encode different problems. In particular, the traveling salesman problem will be examined.

We have already seen the basic way of encoding a problem using a string of zeros and ones, which represent a number in its binary form. We can also use a string of letters, for example ABCDE, or a string of integers, 12345, or just about any string of symbols as long as they can be decoded into something more meaningful.

Imagine we had a problem involving a graph and we needed to encode the adjacency list of the graph. We could create the adjacency matrix, which consists of a one in the i, jth position if there is an arc from node i to node j and a zero otherwise. We could then use the matrix as is or we else could concatenate the rows of the matrix to create one long string of zeros and ones. Notice this time, however, the string is not a binary representation of a number.

This leads us to the first method of encoding a tour of the traveling salesman problem. We do have a graph such as the one described above and we can encode it in the same way, only our matrix will have a one in the i, jth position if there is an arc from node i to node j in the tour and a zero otherwise. For example, the matrix

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

represents the tour that goes from city 1 to city 3, city 3 to city 2 and city 2 to city 1. This encoding is known as matrix representation and is given in [3] and [7].

The traveling salesman problem can also be represented by a string of integers in two different ways. The first (given in [9], [3], [2] and [7]) is by the string

$$v = a_1 a_2 \dots a_n$$

which implies that the tour goes from a_1 to a_2 to a_3, etc and from a_n back to a_1. Notice that the strings $v_1 = 1234$ and $v_2 = 2341$ are equivalent in this representation.

The second way to represent the traveling salesman problem is with cycle notation ([7]), with an integer string

$$v = b_1 b_2 \dots b_n$$

where the tour goes from city i to city b_i. That is, the string $v_1 = 3421$ means that the tour goes from city 1 to city 3, city 3 to city 2, city 2 to city 4 and city 4 to city 1. Note that not every possible string here represents a legal tour, where a legal tour is a tour that goes to every city exactly once and returns to the first city. It is possible for us to have a string that represents disjoint cycles, for example, $v_2 = 3412$ implies that we go from city 1 to city 3 and back to city 1 and from city 2 to city 4 and back to city 2.

2.5 Crossover

Several crossover methods have been developed for the traveling salesman problem. In this section we describe several of them. We shall compare these methods in chapter 4.

We start by looking at partially matched crossover (PMX) ([4], [1], [2] and [7]). Recall the two-point crossover and assume we were to use this with the integer representation defined for the traveling salesman problem in section 2.4. If we performed a two-point crossover on the chromosomes

$$\begin{aligned} v_1 &= 1234 \mid 567 \mid 8 \\ v_2 &= 8521 \mid 364 \mid 7 \end{aligned}$$

we would get

$$\begin{aligned} v_1' &= 1234 \mid 364 \mid 8 \\ v_2' &= 8521 \mid 567 \mid 7 \end{aligned}$$

which are obviously illegal because v_1' does not visit cities 5 or 7 and visits cities 4 and 3 twice. Similarly v_2' does not visit cities 4 or 3 and visits cities 5 and 7 twice. PMX fixes this problem by noting that we made the swaps $3 \leftrightarrow 5$, $6 \leftrightarrow 6$ and $4 \leftrightarrow 7$ and then repeating these swaps on the genes outside the crossover points, giving us

$$\begin{aligned} v_1'' &= 12573648 \\ v_2'' &= 83215674 \end{aligned}$$

In other words, we made the swaps, $3 \leftrightarrow 5, 6 \leftrightarrow 6, 4 \leftrightarrow 7$ and the other elements stayed the same. v_1'' and v_2'' still consist of parts from both the parents v_1 and v_2 and are now both legal.

This crossover would make more sense when used with the cycle representation, since in this case it would preserve more of the structure from the parents. If, as in our example, we used the first integer representation, the order that the cities were visited would have changed greatly from the parents to the children - only a few of the same edges would have been kept. With cycle notation a lot more of the edges would have been transfered. However, if we use this crossover routine with cycle representation we do not necessarily get a legal tour as a result. We would need to devise a repair routine to create a legal tour from the solution that the crossover gives us, by changing as little as possible in order to keep a similar structure.

Cycle crossover (CX) ([4], [2] and [7]) works in a very different way. First of all, this crossover can only be used with the first representation we defined, that is, the chromosome $v = 1234$ implies that we go from city 1 to city 2 to city 3 to city 4. This time we do not pick a crossover point at all. We choose the first gene from one of the parents

$$\begin{aligned} v_1 &= 12345678 \\ v_2 &= 85213647 \end{aligned}$$

say we pick 1 from v_1

$$v_1' = 1 - - - - - - -$$

We must pick every element from one of the parents and place it in the position it was previously in. Since the first position is occupied by 1, the number 8 from v_2 cannot go there. So we must now pick the 8 from v_1.

$$v_1' = 1 - - - - - - 8$$

This forces us to put the 7 in position 7 and the 4 in position 4, as in v_1.

$$v_1' = 1 - - 4 - - 78$$

Since the same set of positions is occupied by 1, 4, 7, 8 in v_1 and v_2, we finish by filling in the blank positions with the elements of those positions in v_2. Thus

$$v_1' = 15243678$$

and we get v_2' from the complement of v_1'

$$v_2' = 82315647$$

This process ensures that each chromosome is legal. Notice that it is possible for us to end up with the offspring being the same as the parents. This is not a problem since it will usually only occur if the parents have high fitnesses, in which case, it could still be a good choice.

Order crossover (OX) ([2] and [7]) is more like PMX in that we choose two crossover points and crossover the genes between the two points. However instead of repairing the chromosome by swapping the repeats of each node also, we simply rearrange the rest of the genes to give a legal tour. With the chromosomes

$$\begin{aligned} v_1 &= 135 \mid 762 \mid 48 \\ v_2 &= 563 \mid 821 \mid 47 \end{aligned}$$

we would start by switching the genes between the two crossover points.

$$\begin{aligned} v_1' &= - - - \mid 821 \mid - - \\ v_2' &= - - - \mid 762 \mid -- \end{aligned}$$

We then write down the genes from each parent chromosome starting from the second crossover point.

$$\begin{aligned} v_1 &: \; 48135762 \\ v_2 &: \; 47563821 \end{aligned}$$

then the genes that were between the crossover points are deleted. That is, we would delete 8, 2 and 1 from the v_1 list and 7, 6 and 2 from the v_2 list to give

$$\begin{aligned} v_1 &: \; 43576 \\ v_2 &: \; 45381 \end{aligned}$$

which are then replaced into the child chromosomes, starting at the second crossover point.

$$\begin{aligned} v_1' &= 57682143 \\ v_2' &= 38176245 \end{aligned}$$

Next we consider matrix crossover (MX) ([7] and [3]). For this we have a matrix representation where the element i, j is 1 if there is an edge from node i to node j and 0 otherwise. Matrix crossover is the same as one- or two-point crossover. If we have the matrices

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

we choose the crossover points after the first column and after the second column and crossover the columns to give

$$A' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$B' = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

We now have multiple $1'$s in some rows and some rows without any $1'$s at all. We fix this by moving one of the $1'$s from the row with the multiples to a row without any 1′s. We choose which 1 to move randomly.

$$A'' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$B'' = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Now notice in A'' we have $a \rightarrow a$ and $b \rightarrow c \rightarrow b$. So we have two different cycles. We can fix this by cutting and reconnecting the cycles. Obviously we would cut the edge from a to a and one of the edges between b and c and connect a to b and a to c. When we have a choice as to which nodes we connect (our example was small enough so that we do not have a choice) we choose the ones that exist in one of the parents to try to maintain the structure as much as possible.

Modified order crossover (MOX) ([9]) is similar to order crossover. We randomly choose one crossover point in the parents and as usual, leave the genes before the crossover point as they are. We then reorder the genes after the crossover point in the order that they appear in the second parent chromosome. If we have

$$\begin{aligned} v_1 &= 123 \mid 456 \\ v_2 &= 364 \mid 215 \end{aligned}$$

we would get

$$\begin{aligned} v_1' &= 123 \mid 645 \\ v_2' &= 364 \mid 125 \end{aligned}$$

The crossovers explored so far concentrate on the position of the city in the tour whereas it is really the edges that are the most important part of the traveling salesman's tour, since they define the costs. So what we really want is to deal with edges rather than the positions of each city.

Grefenstette(1981, cited in [4]) has devised a crossover routine which picks each node from one of those which is incident to the current node in one of the parents. We do this by creating an edge list for each node. The chromosomes

$$\begin{aligned} v_1 &= 123456 \\ v_2 &= 364215 \end{aligned}$$

have edge list

$$\begin{aligned} \text{node } 1: &\quad 256 \\ \text{node } 2: &\quad 134 \\ \text{node } 3: &\quad 2456 \\ \text{node } 4: &\quad 2356 \\ \text{node } 5: &\quad 1346 \\ \text{node } 6: &\quad 1345 \end{aligned}$$

We first choose one of the initial nodes from one of the parents, i.e., 1 or 3 in this example. We choose the one that has the least number of incident nodes, or if they have the same number we randomly choose one. We then consider the nodes incident to node 1 since this is the node we first chose. Again we choose the node with the least number of previously unchosen incident nodes. So we choose node 2. We continue this process of considering nodes which have not previously been selected. If we encounter a situation in which we cannot choose a node that has not previously been selected we randomly choose a previously unselected node. This means that we will get a node which is not incident to our current node in one of the parents, but unfortunately this is unavoidable. So our parent chromosomes could give the offspring

$$v_1' = 124365$$

Notice that we were successful in being able to choose nodes that were incident in one of the parents at all times. We also only get one offspring from this crossover so we need to do twice as many crossovers to create the new generation.

We also have crossover operators that use heuristic information. The heuristic crossover ([4]) chooses a random node to start at and then considers the two edges leaving the current node in the parent chromosomes and picks the shortest edge that does not introduce a cycle. If both edges introduce a cycle we choose a random edge that does not do so.

2.6 Mutation

First we will look at the 2-opt operator ([4]). We randomly select two edges (a, b) and (c, d) from our tour and check if we can connect these four nodes in a different manner that will give us a lower cost. To do this we check if

$$c_{ab} + c_{cd} > c_{ac} + c_{db}$$

If this is the case we replace the edges (a, b) and (c, d) with the edges (a, c) and (d, b). Note that we assume that a, b, c and d appear in that specific order in the tour even if b and c are not connected.

We also have a 3-opt operator ([4]) which looks at three random edges instead of two. If we have edges (a, b), (c, d) and (e, f), we check if

$$c_{ab} + c_{cd} + c_{ef} > c_{ac} + c_{be} + c_{df}$$

If it is we replace (a, b), (c, d) and (e, f) with the edges (a, c), (b, e) and (d, f).

The Or-opt operator ([4]) is similar to the 3-opt. We randomly choose a set of connected nodes and check if this string can be inserted between two other connected nodes to give us a reduced cost. We can calculate this by finding the total cost of the edges being inserted and the total cost of the edges being removed. If the cost of the edges being removed is greater than the cost of those being inserted the switch is made.

Another three mutation operators (given in [7]) are insertion where we randomly select a city and insert it in a random place. Displacement is where we select a subtour and insert it in a random place. We also have reciprocal exchange where we choose two random cities and swap them.