A STUDY OF TRAVELLING SALESMAN PROBLEM USING REINFORCEMENT LEARNING OVER GENETIC ALGORITHM

B. Biswas[1]
Dept. of Computer Applications, Haldia Institute of Technology, Haldia
Email: bipasm@gmail.com
A. Mitra[2]**(Corresponding author)**
Dept. of Computer Applications, Haldia Institute of Technology, Haldia
Email: apratim777@yahoo.com
S. Sengupta[3]**(Corresponding author)**
Dept. of Computer Applications, Haldia Institute of Technology, Haldia
Email: satadru.sengupta@gmail.com

Abstract: This paper represents the applications of Genetic Algorithm (GA) to solve a Travelling Salesman problem (TSP). TSP is a simple to describe and mathematically well characterized problem but it is quite difficult to solve. This is a NP-hard type problem i.e. this problem is hard as the hardest problem in NP-complete space. We present the Crossover and Mutation operators, sorting of the solutions to calculate the best optimal solutions. Previously, a numerical illustration was used to signify the model with the techniques. This paper employs Reinforcement Learning to solve the Traveling Salesman problem in the mean of Genetic Algorithm. The technique proposes a model (actions, states, reinforcements).

Keywords: Travelling Salesman problem (TSP), Genetic Algorithm (GA), Reinforcement Learning

1. INTRODUCTION

The Travelling Salesman problem is a classical combinatorial / optimization problem. This problem is like: A salesman travels n cities like that a salesman starts his journey by selecting any random city among having(n) cities, traverses the remaining cities one by one and returns again to the first city from where the journey was started. So the journey of the Sales man chalks down a complete tour that consists of all the cities. Now the resultant TSP is to find the smallest path with cheapest cost. The total approach is represented by a complete weighted graph G=(V,E) where V implies a vertex sets(cities) and E implies sets of edges fully connected with the nodes. Each edge (i,j) ϵE is assigned to a weight d_{ij}which represents the distance between i and j. Formally the goal is to find a Hamiltonian tour of minimal length on a fully connected graph. The problem can have number of feasible solutions but the outcome that will give best result in terms of space and time will be represented as the optimal solution or prove that there is no feasible solution.

2.Used Methods

Here, the cities and distance between pair of cities are shown as the matrix, which is Adjacency. Now, we need to deduct the shortest possible rout while each city is traversed exactly once and after traversal all of the cities the starting city is to be reached. This is actually the Travelling Salesman Problem.

There are many approaches to solve this problem. We can solve the problem by mainly two means of approaches. We can go for an optimal approach to found the length otherwise the Held-Karp lower bound approach also can be taken.Several numbers of algorithms are eligible to get the solution of the dynamic programming like TSP. If number of city is very large the desired solution is not feasible. The feasibility of algorithm is found by Held-Karp lower bound. Usually, HK lower bound shows 0.8% below the optimal path. Several types of mean can be taken to understand the TSP problem. Simply we can use Geometry to visualise and solve when this can be as justified as finding the merely the good path. This can be used for improving the path length, removing the acute angles or crossed edges from the optimal tour. The complex methods may be like: the cutting-plane, the primal approach, the branch-and-bounding method, Dynamic Programming, the Branch-and-Cut method, linear programming etc. We will show these several methodical approaches in the following sections

3. Approximation Methods: The Approximation methods are the chosen one to find a tour. The level of efficiency of tour is measured by the computational time, the simplest calculation accuracy, which may be measured in a best or worst-case happening ratio, depending on the algorithm used.

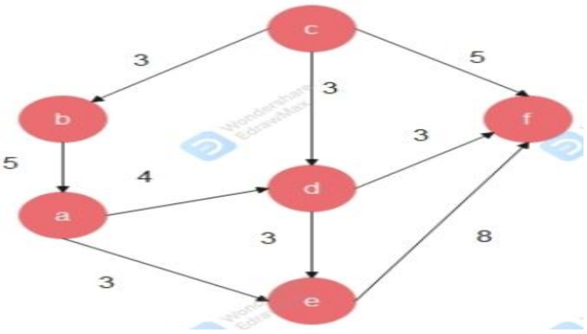


Fig 1

We have the above graph and we will apply our well known TSP problem using some approaches.

3.1. Nearest neighbour: The simplest approximation method is the Nearest Neighbour Algorithm. Essentially a random city is selected as starting node, then the nearest city, which is still left unvisited, or the prior city with the lowest cost, is selected until there is no unvisited city left, where the traversal path goes to the starting city and is the tour is complete. This step-by-step process is illustrated in the example below. In this approach the tour that is found works out nicely, but then if the salesman needs to traverse large distances to reach all of the cities there is the chance to carelessly miss it in the beginning. This algorithm in worst case gives $1 + \frac{\log n}{2}$times cost of an optimal path, where n means the how many cities. The computational time of the the nearest neighbor algorithm is O(n ^{2}) , whence there are n(n−1) ^{2} edges found in a complete graph (Traveling Salesman Problems are usually represented as complete graphs). Expanding the given number of edges we get $\frac{n2}{2} - \frac{n}{2}$. As the Big O notation can consider the most significant only, the computational time of the nearest neighbor algorithm is O(n ^{2}).

Repetitive Nearest Neighbour Algorithm

Step 1.Input a node and set the Nearest Neighbour Algorithm with the node that is set a START.

Step 2. Repeat Nearest Neighbour Algorithm for each node used in the graphical presentation.

Step 3.Set the best of all Hamilton circuits for Steps 1 and 2.

Step 4. Print the solution by using the current node a START.

This algorithm has disadvantages and depends on the distribution of cities so the optimal solution may or may not reached.

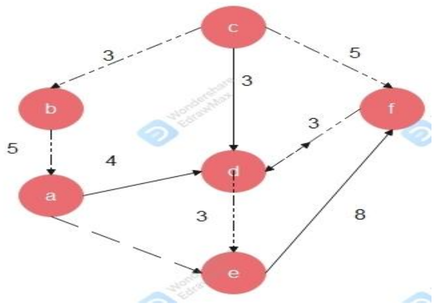


Fig 2

The reqired Hamiltonian Circuit is formed from Nearest Neighbor algorithm applyingFig 1 . The cheapest path costs A-E-D-F-C-B-A i.e. cheapest cost is 22.

3.2. Greedy algorithm: The Greedy Algorithm creates many subpaths by adding the shortest available path segment. The traversal process is shown in the following steps.

Step 1.Set all the edge with their weights according to increasing manner.

Step 2. Create the list by adding edges one by one in the covered path list if and only if the considering edge does not violates the undergoing rules:

(a) The selected node must have degree of at most 2 in the considering path set.

(b) Any cycle must not be formed unless the number of edges are equalling the number of nodes in the graph i.e. the traversal is complete.

This process is illustrated in the example below. This algorithm can give worst computation time as $\frac{1}{2}+\frac{\log(n)}{2}$ times comparing with an optimal tour, (n=number of cities).The computationaltime of this algorithm is $O(n^2\log_2(n))$. The optimal path is a-e-d-f-c-b-a. The cost is: 22, i.e same as Nearest Neighbour.

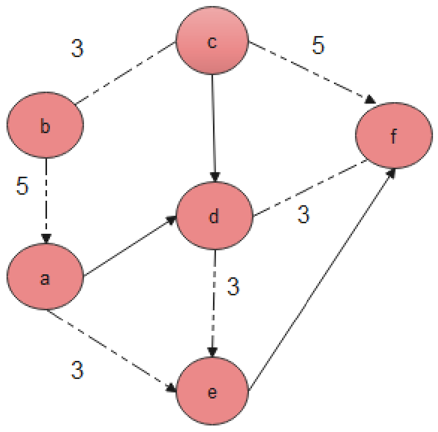


Fig 3

3.3. Insertion Heuristics:

An insertion algorithmcreates a sub-tour on ***k*** nodes at iteration ***k*** and it determines which of the remaining ***n-k*** nodes can be inserted to the next sub-tour, i.e. the selection step andbetween which nodes the sub-tour can be inserted (the insertion step).

Nearest Insertion

Step 1. Select a sub-graph which contains only node i.

Step 2. Select node **r**where $\mathbf{C_{ir}}$is minimal and promote a sub-tour **i-r-i**

Step 3. Selection step:From the selected sub-tour get node **r** which is not in the sub-tour, that is closest to any node **j** containing in the sub-tour; i.e. with minimal $\mathbf{C_{rj}}$

Step 4. Insertion step: Find the joining arc **(i, j)** in the sub-tour ,minimizing$\mathbf{C_{ir} + C_{rj} - C_{ij}}$. Now insert the selected **r** in between **i** and **j**.

Step 5. If all the nodes are added to the tour, then stop.

Else go to step 3

Worst Case occurrence: If it happens like the resultant value after dividing length of nearest insertion tour by length of optimal tour is less than or equal to 2 the algorithm gives worst time complexity.

$$\frac{Length\_of\_nearest\_insertion\_tour}{Length\_of\_optimal\_tour} <= 2$$

Number of Computations: The average time complexity of nearest insertion algorithm is algorithm is$O(n^2)$.

If we follow this algorithm for fig.1 we get optimal solution as 22,i.e. same as Greedy algorithm.

3.4. Christofides

Dr.NicosChristofides, a professor at Imperial College London, combined a minimum cost spanning tree, a tree that spans the entire set of cities, with a perfect matching, a set of edges that meet each of the odd vertices exactly once. An odd vertex is a vertex that has an odd degree in the spanning tree. The steps are as follows:

Step 1. Find a minimal spanning tree that covers the graph.

Step 2. Connect each odd vertex in the tree with exactly another one odd vertex to create a perfect matching using Edmonds' matching algorithm.

Step 3. Build anEulerian cycle from the resulting graph, and then traverse the cycle which is the shortest path.

Actually this is the shortest path because all the graphs we are selecting satisfy the triangle inequality, so the weights of the edges must satisfy $e(v, u) + e(v, w) \geq e(u, w)$, for all of the combinations of the pair of vertices v, u, and w. The resulting cycle is the desired Travelling Salesman tour. In comparison to the other methods thus far discussed, the Christofides' algorithm has the best worst case scenario with a guarantee of at most 1.5 times in comparison with an optimal solution. The algorithm's computational time is $O(n^3)$.

3.5. The Brute-Force Approach: The Brute Force approach, also known as the Naive Approach, calculates and compares all the possible permutations of routes or paths to determine the shortest unique solution. To solve the TSP using the Brute-Force algorithm, we have to calculate the total number of tours and then draw and enlist all the possible tours. By calculating the distance of each tour and then choosing the shortest one we get the optimal solution.

3.6. The Branch and Bound Method: This algorithm breaks a problem into several sub-problems. It's a system which solves a series of sub-problems, where each of the sub problems must have several possible solutions.The solution that is selected for one problem, caneffect the possible solutions of the subsequent sub-problems. To solve the TSP using the Branch and Bound method, we must use the following steps

Step 1. Choose a starting node.

Step 2. Set bound to a very large value (let's say infinity).

Step 3. Select the cheapest arc between the unvisited and current node and then add the distance to the current distance.

Step 4.Repeat the process while the current distance is less than the bound and optimal solution is found.

Step 5. Add the distance so that the bound will be equal to the current distance. Repeat until all the arcs have been covered.

TSP is a minimization problem; we consider f(x) as a fitness function, where f(x) decides cost (or value) of the tour between cities (nodes).

In the next section we expose the different optimization techniques that will be used to solve TSP.

4. Used Methods

4.1. Ant Colony optimization: The Ant Colony Optimization is a probabilistic technique which is connected to artificial intelligence technology and it is used to design meta-heuristic algorithms to solve combinatorial optimization problem. The algorithm called Ant System (AS), it was the first algorithm and was applied to TSP. Dorigo, used the concept of ant colony algorithms based on the mathematical models of ant colony's behaviors on the TSP problem and acquired the desired results. ACO is a technique inspired by the way of ants searching food. The secreted pheromone from the ants helps them to find the shortest path between food sources and nests.

The process of the ACO to solve TSP is described here.

Tour Construction: Initially, the artificial agents acting like ants are distributed to n cities randomly and determines the other agent's positions on different cities with initial value $T_{ij}(0)$, where each ant k applies a probabilistic rule in each step while the probabilistic rule helps the ants which is stepped in city i to decide to visit the next neighboring city j. The Probabilistic rule is as followed:

$$p_{ij}^{k}(t) = \left\{ \frac{white\left[T_{ij}(t)\right]^{\infty}\left[\mu_{ij}(t)\right]^{\beta}}{white0\sum_{k}\left[T_{ik}(t)^{\infty}\left[\mu_{ik}(t)\right]^{\beta}\right]} \right\} \text{ if } j\epsilon J_k(i)$$

Where p_{ij}^{kt} is the probability of the ant passing from node i to node j, the τ_{ijt} is the pheromone value between any node i and j, μ_{ij}(t) is the heuristic value between nodes i and j at t moment, $\mu_{ij}= 1/d_{ij}$, as d_{ij} is the distance between nodes i and j and $\infty$and β are two adjustable positive parameters that control the relative weights of the edge of pheromone trails and of the heuristic visibility. After n iterations of this process, every ant completes a tour and a feasible solution of TSP is represented by the path of ants which visited all the nodes that are appeared as a sequence including all serial numbers of cities. For every iteration, all of the ants'act should be estimated by the mean of the objective function. Then, two high-quality ants or agents are chosen for the pheromone deposition on their edges: the first is the best ant in the current iteration, and the second is the global best ant found so far. Computation of the optimal path is the stopping criteria of the algorithm which iterates a certain number of iterations, after the pheromone trail updating process and then the k ants have travelled through all the cities and the next iteration is (t + 1) that will update the distances between cities.

4.2. Particle Swarm Optimization (PSO) algorithm: Particle Swarm Optimization algorithm was first proposed by Kennedy and Eberhart in 1995. The used to believe that a school of fish or a flock of birds that moves in a group "can profit from the experience of all other members". That means while a bird is flying and searching randomly for food, for instance, all the birds in the flock can share their findings, the way of their findings and this objectives of them used to help the entire flock get the food in short time and using much smaller effort.Like the movement of a flock of birds, we can also imagine that each bird is to help us to find the optimal solution in a high-dimensional solution space and the best solution is found by the flock is the best solution in the space. This is a **heuristic solution** because we can never prove the real **global optimal** solution can be found and it is usually not. However, we often find that the solution found by PSO is quite close to the global optimal.

The **PSO** algorithm can be used to solve our very well-known Travelling Salesman Problem. Let we are using a graph G = (V,E), where V = {1,...,n} and E = {1,...,m}, and costs, cij, associated with each edge joining vertices i and j, the TSP consists in finding the minimal total length Hamiltonian cycle of G. The length is calculated by the summation of the costs of the edges in the considered cycle. If for all pairs of nodes {i,j}, the costs cij and cji are equal then the problem is said to be symmetric, otherwise it is said to be asymmetric.

A general approach of a particle swarm optimization algorithm is shown below. Initially, a population of the particles is generated. When all particles are evaluated and, then, $pbest_p$ is replaced by x_p, p's position. The best position is achieved so far by any of the p's neighbors is set to $gbest_p$. Finally, the velocities and the positions of each particle are updated. The procedure compute_velocity( ) receives three inputs. This is done to show that, in general, p's position, x_p, $pbest_p$ and $gbest_p$ are used to update p's velocity, v_p. The process is repeated until some stopping condition is satisfied.

```
procedure PSO
 Initialize a population of particles
do
        for each particle p with position $x_p$ do
                if ($x_p$ is better than $pbest_p$) then
                pbest$_p$ ← $x_p$
                end_if
        end_for
        Define $gbest_p$ as the best position found so far by any of p's neighbors
        for each particle p
do
        $v_p$ ← Compute_velocity($x_p$, $pbest_p$, $gbest_p$)
        $x_p$ ← update_ position($x_p$, $v_p$)
end_for
while (a stop criterion is not satisfied)
```

Kennedy &Eberhart (1995) suggested equations (1) and (2) to update the swarm particle's velocities and the positions, respectively. In these equations, $x_p(t)$ and $v_p(t)$ are the particle's position and velocity at instant t, $pbest_p(t)$ is the best position and the particle was achieved up to instant t, $gbest_p(t)$ is the best position that any of p's neighbors achieved up to instant t, c_1 is a cognitive coefficient that quantifies how much the particle trusts its experience, c_2 is a social coefficient that quantifies how much the particle trusts its best neighbor, $rand_1$ and $rand_2$ are random numbers.

$$v_p(t) = v_p(t-1) + c_1.rand_1.(pbest_p(t-1) - x_p(t-1)) + c_2.rand_2 .(gbest_p(t-1) - x_p(t-1)) \quad (1)$$

$$x_p(t) = x_p(t-1) + v_p(t) \quad (2)$$

Goldbarg et al. (2006a) applied Swarm optimization on TSP. They implemented two different versions of the PSO algorithm defined by the two local search procedures utilized to implement v_1. In the first one, a local search procedure based on the inversion neighborhood is used. The LinKernighan (Lin & Kernighan, 1973) neighborhood is used in the second version. Both of the versions are implemented with the path-relinking procedure. The particles' positions are represented as the permutations of the |N| cities. In the inversion neighborhood, it is given a sequence $x_1 = (n_1, \ldots, n_i, n_{i+1},\ldots, n_{j-1}, n_j, \ldots, n_{|N|})$ and two indices i and j, the sequence x_2 is x_1's neighbor if $x_2 = (n_1, \ldots, n_j, n_{j-1},\ldots, n_{i+1}, n_i, \ldots, n_{|N|})$. The difference between indices i and j varies from 1 to |N|-1. When v_1 is applied to a particle p, then the local search procedure starts inverting the sequences of any two elements in p's position, then the sequences of the three elements are inverted, and so on. The Lin-Kernighan neighborhood is a recognized efficient improvement method for the TSP.

4.3. Genetic Algorithm: Genetic Algorithm (GA) is a search-based optimization technique based on the principles of **Genetics and Natural Selection**. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning.The optimization refers to find the values of inputs in such a way that we get the best output values. The definition of "best" varies from problem to problem, but in mathematical terms, it refers to maximizing or minimizing one or more objective functions, by varying the input parameters.

The set of all possible solutions or values which the inputs can take to make up the search space. In this search space, there lies a point or a set of points which gives the optimal solution. The aim of optimization is to find that point or set of points in the search space. The Genetic Algorithms (GAs) are search based algorithms which is based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as **Evolutionary Computation**.GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.

In GAs, we have a **pool or a population of possible solutions** to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more "fitter" individuals. This is in line with the Darwinian Theory of "Survival of the Fittest". In this way we keep "evolving" better individuals or solutions over generations, till we reach a stopping criterion. Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well. Genetic Algorithms, it is essential to be familiar with some basic terminology which will be used in this paper.

- **Population** –The subset of all of the possible (encoded) solutions to the given problem.
- **Chromosomes** − A chromosome is one solution to the given problem.
- **Gene** − A gene is the element position of a chromosome.
- **Allele** –The value a gene takes for a particular chromosome.

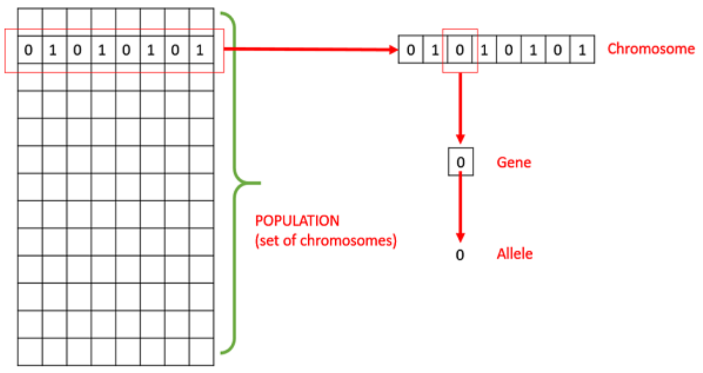


Fig 4

- **Genotype** –The Genotype is the population in the computation space. In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a computing system.
- **Phenotype** − Phenotype is the population in real world solution space in which solutions are represented in a way they are represented in real world situations.
- **Decoding and Encoding** –The **phenotype and genotype** spaces are the same. However, in most of the cases, the phenotype and genotype spaces are different. The Decoding is the process of transforming a solution from the genotype to the phenotype space, while the encoding is a process of

transforming from the phenotype to genotype space. Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation.

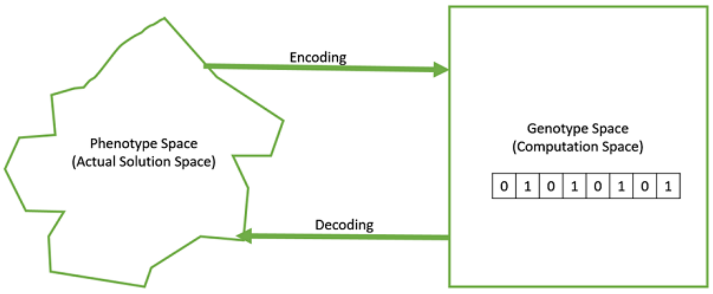


Fig 5

- **Fitness Function** − A fitness function is a function which takes the solution as input and produces the output. In some cases, the fitness function and the objective function may be the same, while in others it might be different.
- **Genetic Operators** –The operators alter the genetic composition of the offspring. These include crossover, mutation, selection, etc.The basic structure of a GA is as follows –

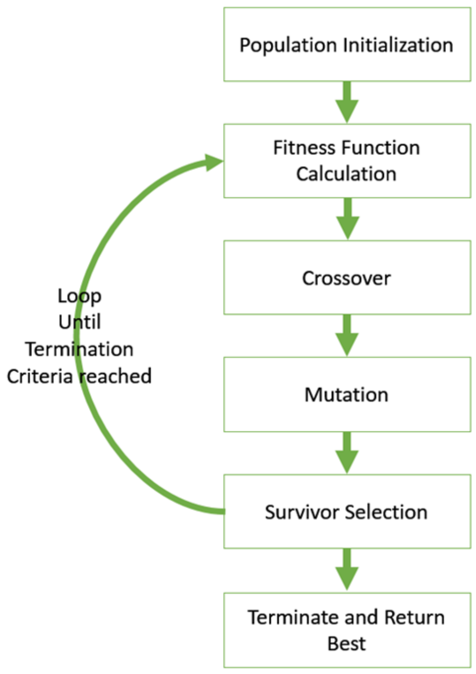

Population Initialization
Fitness Function Calculation
Crossover
Mutation
Survivor Selection
Terminate and Return Best
Loop Until Termination Criteria reached

Fig 6

Algorithm(GA)

```
1.  Set Current=Initial population;
2.  Initiate New-generation=Ø
3.  Repeat
        3.1. First=Randomly selected individual  from current according to  Fitness function ;
        3.2. Second= Randomly selected individual  from current according to  Fitness function ;
        3.3. New_individual=Crossover (first, second);
        3.4. If(fitness(New_individual<=threshold)
            Set Mutation(New_individual);
        3.5. New generation=(New generation U  New_individual)
4.  Set Current=New generation;
5.  Repeat step 3 until some individual from current fits enough;
6.  Return(The best individual from current according to fitness)
```

Actually chromosome is a data structure, usually a string that represents a candidate solution.
The population is a collection of chromosomes.
Each chromosome has a numerical fitness, which indicates the quality of chromosome's fitness. There is an example to make understand about how the Genetic algorithm works is shown below:

A. Start from the representation of chromosomes.

Encode solutions as binary strings: sequences of 1's and 0's, where the digit at each position represents the value of some aspect of the solution.

Encoding of chromosomes:

Chromosome 1	1101100100111100
Chromosome 2	1101111000011110

B. Fitness function
The GAs are used for maximization problem. For the maximization problem the fitness function is same as the objective function. But, for minimization problem, one way of defining a 'fitness function' is as F(x)=1/f(x), where *f*(x) is the objective function.
C. Selection:

There are many methods how to select the best chromosomes, for example roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and some others.

C.(i) Roulette Wheel Selection:Conceptually, this can be represented as a game of roulette - each individual gets a slice of the wheel, but more fit ones get larger slices than less fit ones. Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. Imagine a **roulette wheel** where are placed all chromosomes in the population, everyone has its place big accordingly to its fitness function, like on the following picture.

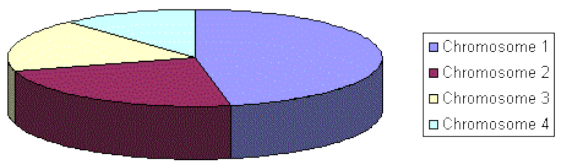


Fig 7

Chromosome with bigger fitness will be selected more times. This can be simulated by following algorithm.

1. **[Sum]** Calculate sum of all chromosome fitnesses in population - sum ***S***.
2. **[Select]** Generate random number from interval ***(0,S)*** - ***r***.
3. **[Loop]** Go through the population and sum fitnesses from ***0*** - sum ***s***. When the sum ***s*** is greater than ***r***, stop and return the chromosome where you are.

Of course, step **1** is performed only once for each population.

C. (ii) Rank selection:

Each individual in the population is assigned a numerical rank based on fitness, and selection is based on this ranking. The previous selection will have problems when the finesses differ very much. For example, if the best chromosome fitness is 90% of the entire roulette wheel then the other chromosomes will have very few chances to be selected. Rank selection first ranks the population and then every chromosome receives fitness from this ranking. The worst will have fitness ***1***, second worst ***2*** etc. and the best will have fitness ***N*** (number of chromosomes in population). You can see in following picture, how the situation changes after changing fitness to order number.

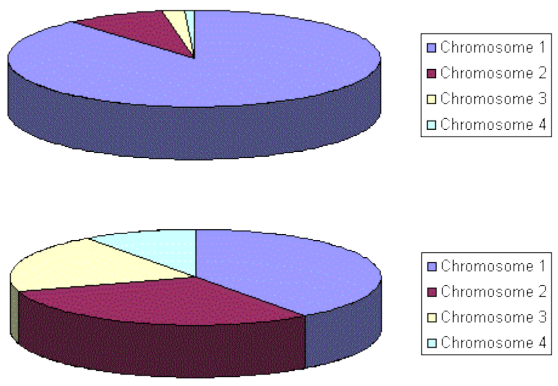


Fig 8

C. (iii) Elitist selection:

The fit members of each generation are guaranteed to be selected.

D. Reproduction

Once selection has chosen fit individuals, they must be randomly altered in hopes of improving their fitness for the next generation.

1. Methods of Reproduction:-Select parents for the mating pool
   (Size of mating pool = population size)
2. Shuffle the mating pool
3. For each consecutive pair apply crossover with probability p_c , otherwise copy parents
4. For each offspring apply mutation (bit-flip with probability p_m independently for each bit)
5. Replace the whole population with resulting offspring.

Crossover and Mutation

a. Crossover operation:

Crossover can look like this (│ is the crossover point)

Chromosome 1	1101100100111100
Chromosome 2	1101111000011110
Offspring 1	11011 │ 11000011110
Offspring 2	11011 │ 00100111100

There are other ways to make crossover; for example we can choose more crossover points. Specific crossover for specific problem can improve performance of the genetic algorithm.

b. Mutation operations:

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible. Mutation is an important part of the genetic search as helps to prevent the population from stagnating at any local optima. Mutation occurs during evolution according to a user-definable mutation probability. If it is set to high, the search will turn into a primitive random search. There are different types of mutation.

Original Offspring 1	1101111000011110

Bit Flip -A mutation operator that simply inverts the value of the chosen gene (0 goes to 1 and 1 goes to 0). This mutation operator can only be used for binary genes. There are other ways to make Mutation operation. Such as,

Non-Uniform-A mutation operator that increases the probability that the amount of the mutation will be close to 0 as the generation number increases. This mutation operator keeps the population from stagnating in the early stages of the evolution then allows the genetic algorithm to fine tune the solution in the later stages of evolution. This mutation operator can only be used for integer and float genes.

Uniform-A mutation operator that replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. This mutation operator can only be used for integer and float genes.

Gaussian-A mutation operator that adds a unit Gaussian distributed random value to the chosen gene. The new gene value is clipped if it falls outside of the user-specified lower or upper bounds for that gene. This mutation operator can only be used for integer and float genes.

An example of Bit flip is given below:

Original Offspring 1	1101111000011110
Original Offspring 2	1101100100110110
Mutated Offspring 1	1100111000011110
Mutated Offspring2	1101101100110110

E. Recommendations

Recommendations are often results of empiric studies of GAs that were often performed on binary encoding only.

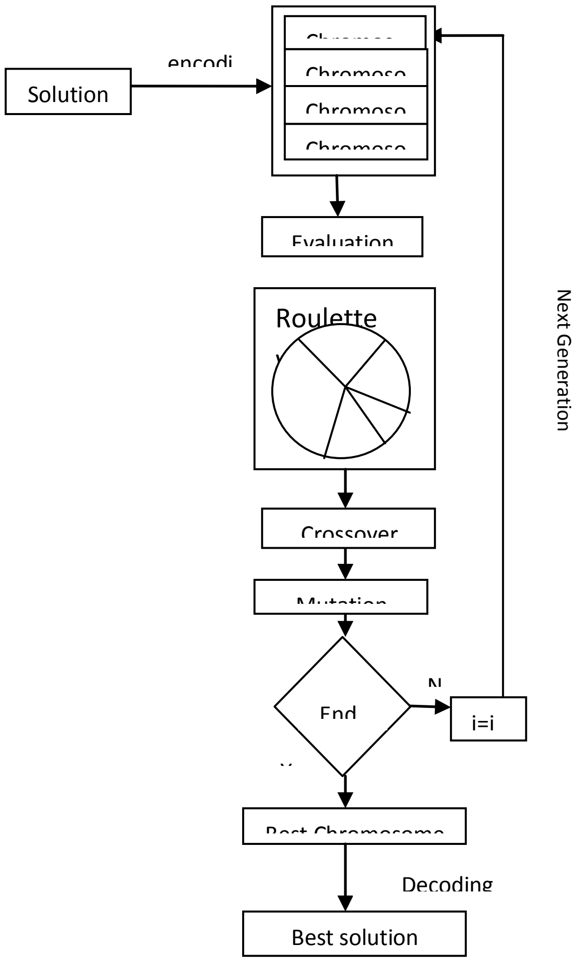
• **Crossover rate**: Crossover rate should be high generally, about **80%-95%**. (However some results show that for some problems crossover rate about 60% is the best.)

• **Mutation rate**: On the other side, mutation rate should be very low. Best rates seems to be about **0.5%-1%**

.•**Population size**: It may be surprising, that very big population size usually does not improve performance of GA (in the sense of speed of finding solution). Some research also shows, that the best population size depends on the **size of encoded string** (chromosomes). It means that if you have chromosomes with 32 bits, the population should be higher than for chromosomes with 16 bits.

• **Selection**: Basic **roulette wheel selection** can be used, but sometimes rank selection can be
better. There are also some more sophisticated methods that change parameters of selection during the run of GA. Basically, these behave similarly like simulated annealing.

Original Offspring 2	1101100100110110
Mutated Offspring 1	1100111000011110
Mutated Offspring2	1101101100110110



Flowchart of Genetic Algorithm Fig:9

Elitism should be used for sure if you do not use other method for saving the best found solution.

Encoding: Encoding depends on the problem and also on the size of instance of the problem.**Crossover and mutation type.**Operators depend on the chosen encoding and on the problem.

5.1. PROBLEM FORMULATION:

Let C be the matrix of shortest distances (dimension *nxn*), where n is the number of nodes of graph G i.e., cities. The elements of matrix C represents the shortest distances between all pairs of nodes (i, j), i, j=1, 2, …,n. The travelling salesman problem can be formulated in the category programming binary, where variables are equal to 0 or 1, depending on the fact whether the route from node i to node j is realized (x_{ij}= 1) or not (x_{ij}=0). Then, the mathematical formulation of TSP is as follows (the idea of this formulation is to assign the numbers 1 through n to the nodes with the extra variables u_i, so that this numbering corresponds to the order of the nodes in the tour. It is obvious that this excludes sub tours, as a sub-tour excluding the node 1 cannot have a feasible

assignment of the corresponding u_i variables). The Euclidean distance d, between any two cities with coordinate (x_1, y_1) and (x_2, y_2) is calculated by:
$d= ((x_1 – x_2)^2+ (y_1-y_2)^2)^{1/2}$

$$\min f = \sum_{i=1}^{m}\sum_{j=1}^{n} C_{ij}\, x_{ij}$$

Subject to

$$\sum_{i=1}^{m} x_{ij} = 1\ ,\ j = 1,2,..n.\ and\ j \neq i$$

$$\sum_{j=1}^{n} x_{ij} = 1\ ,\ i = 1,2,..m.\ and\ i \neq j$$

$$u_i - u_j + n x_{ij} \leq n-1, \qquad i \neq j. and\ i,j = 2,3,\dots n$$
$$x_{ij} \in \{0.1\}, for\ all\ i,j = 1,2,\dots,n.$$

An implicit way of solving the TSP is simply to list all the feasible solutions, evaluate their objective function values and pick out the best. However it is obvious that this “exhaustive search” is grossly inefficient and impracticable because of vast number of possible solutions to the TSP even for problem of moderate size. Since practical applications require solving larger problems, hence emphasis has shifted from the aim of finding exactly optimal solutions to TSP, to the aim of getting, heuristically, “good solutions‟ in reasonable time and “establishing the degree of goodness‟. Genetic algorithm (GA) is one of the best heuristic algorithms that have been used widely to solve the TSP instances. A genetic algorithm can be used to find a solution in much less time. Although it might not find the best solution, itcan find a near perfect solution for a 100 city tour in less than a minute.

5.2.NUMERICAL ILLUSTRATION:

To solve the traveling salesman problem, we need a list of city locations and distances, or cost, between each of them. The following basic steps are used to solve the traveling salesman problem using a GA.

a. First, create a group of many random tours in what is called a population. This algorithm uses a greedy initial population that gives preference to linking cities that are close to each other.
b. Second, pick 2 of the better (shorter) tours parents in thepopulation and combine them to make 2 new child tours. Hopefully, these children tour will be better than either parent.
c. A small percentage of the time, the child tours is mutated. This is done to prevent allthe tours in the population from looking identical.
d. The new child tours are inserted into the population replacing two of the longer tours. The size of the population remains the same.
e. New children tours are repeatedly created until the desired goal is reached.

To apply GA for any optimization problem, one has to think a way for encoding solutions as feasible chromosomes so that the crossovers of feasible chromosomes result in feasible chromosomes. The techniques for encoding solutions vary by problem and, involve a certain amount of art. For the TSP, solution is typically represented by chromosome of length as the number of nodes in the problem.

5. a) Representation:

Representation is an ordered list of city numbers known as an *order-based* GA.

1) London 3) Dunedin 5) Beijing 7) Tokyo

2) Venice 4) Singapore 6) Phoenix 8) Victoria

CityList1 (3 5 7 2 1 6 4 8)

CityList2 (2 5 7 6 8 1 3 4)

5. b) Crossover: Crossover combines inversion and recombination:

* *

Parent1	3	5	7 2 1 6	4	8
Parent2	2	5	7 6 8 1	3	4
Child	5	8	7 2 1 6	3	4

This operator is called the *Order1* crossover.

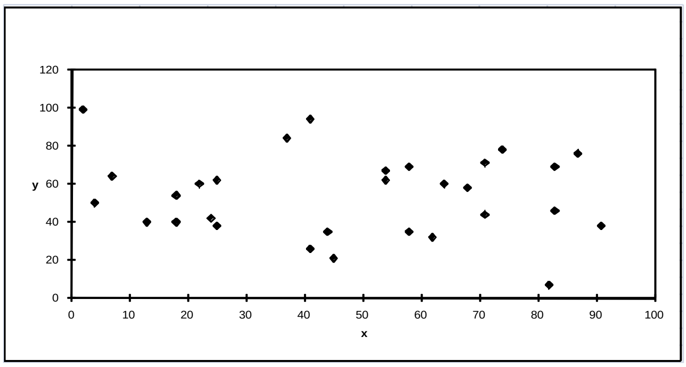
5. c) Mutation : Mutation involves reordering of the list:

* *

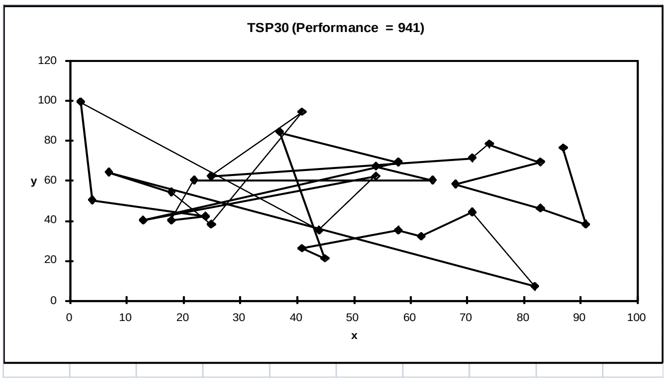
Before	5	8	7	2	1	6	3	4
After	5	8	6	2	1	7	3	4

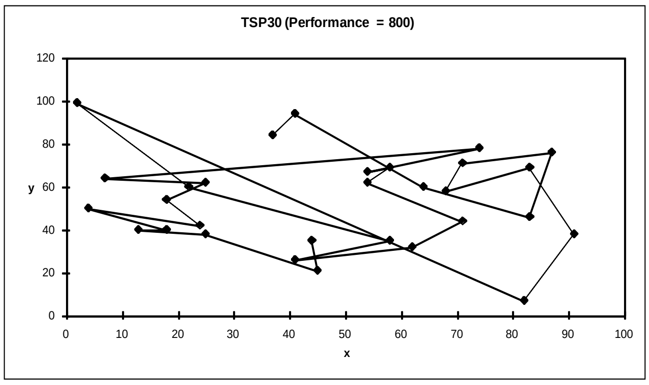
5.d) Sorting:

TSP Example: 30 Cities



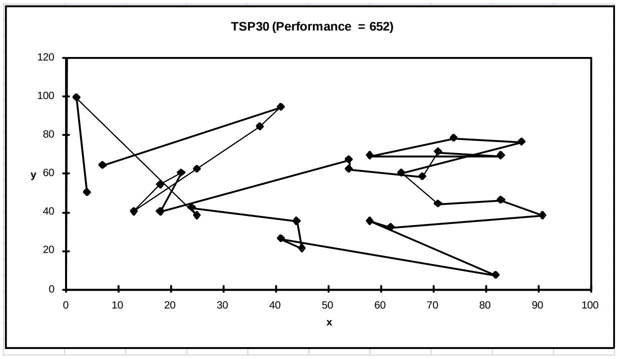
Solution$_i$(Distance=941)



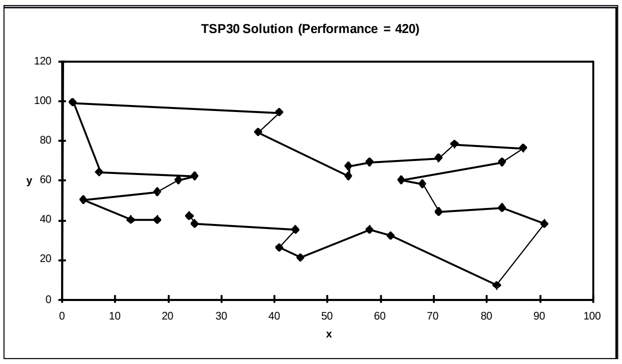


Solution $_k$(Distance = 652)

Solution $_i$(Distance=652)



Best Solution (Distance = 420)



But from this approach it is clear that the most promising nodes that were being elected are elected without knowing whatever they are best suited or not. We can train the collective values and have the knowingly best suited and most promising nodes if we use the learning paradigm.

6.1.Reinforcement learning over GA: Here is an approach of TSP problem that is trying to be solved in the way where the prediction of selecting the most feasible data are being trained. The guideline to apply reinforcement learning over Genetic algorithm after a Numerical approach is applied over, to prevent the identical tour population is followed . If we use a trained prediction policy to get most feasible generation of nodes then the chance of any low estimated generation will almost be nil.

1. Data define the problem using graphs

2. Embed Obtain dense representations of nodes and edges using GNN model

3. Predict Compute probability of nodes/edges belonging to the solution

4. Search Enforce feasibility and constraints through graph search

5. Train Learn prediction policy through imitation (SL) or experience (RL)

The Reinforcement learning is actually the process of Machine Learning. Various software and machines are used to find the best possible solution or path it should take in a specific situation. In supervised learning the training data has the probable key to get the possible output from which we can train the model to give correct answer. In reinforcement learning, there is no such a type of probable output set as answer key but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

In case of GA, we must have a random data set or set of Chromosomes. Then applying crossover and mutation process we can get the best promising chromosome from parent Chromosomes. Here the actual disadvantage is concealed. May be it is a very popular algorithm but there are some basic disadvantages. The very basic disadvantage of Genetic algorithm is its unguided mutation. The mutation operator in GA functions like adding a randomly generated number to a parameter of an individual of the population. This is the only reason of a very slow convergence of genetic algorithm. We can solve this problem by combining it with some machine learning algorithm like RL which perform guided search like Differential evolution. There is a problem of over fitting which can make the performance analysis of GA low.

Over-fitting.

Since we cannot generate all the possible permutations of the set of data we are bound to have over-fitting with our sample data using the simple weighted approach we mentioned. If we do not train the algorithm long enough (generations) then the usability would be low.

Moreover the **Optimization Time**also is so far predicted as very high which is not desirable. If the genetic algorithm has a predefined amount of time during which it can be run, the population size has to be optimized depending on the amount of time instead of the number of generations. If the population size is set too low the genetic algorithm will have too few possible ways to alter new individuals so the fitness will be low. If the population size is set too high the genetic algorithm will take longer time to run each generation, therefore the genetic algorithm does not have so many generations to alter the population, so the fitness will be low.

6.2. The Reinforcement Learning over TSP: There are a few reasons of thinking about RL which offers some unique value for any optimization problem.

1. RL performs well in high-dimensional spaces, especially in those cases if an approximate solution to a complex problem may have value than optimal solution of a simpler problem.

2. RL can perform well in partially observed environments. If there are aspects of a problem we don't know about and therefore can't model, which is often the scenario in the real-world (and we can pretend is the case with these problems), RL's ability to deal with the scenario is justifiable.

3. There are clever strategies we can use to solve versions of TSP, Knapsack, Newsvendor. RL might surprise us.

In the TSP problem we will use let we have a 5x5 grid. All possible paths from node to node traversal are generated where nodes are fixed, and are invariant (non-random). The objective is to visit node at least once ,and return to the starting node from where the agent or salesman started journey.

States: At each time step, our agent is aware of the following information:

1. For the nodes
   Location_city(x,y coordinates)
2. For the salesman
   Location_sales(x,y coordinates)
   Has salesman started from a node? (yes/no)
3. For each order dispatch to each city:
   3.1. Location_city(x,y coordinates)
   3.2. Status (Delivered or Not Delivered)
   3.3. Time (Time taken to deliver reach order – incrementing until delivered)
4. Miscellaneous
   4.1.Time since start of journey.
   4.2. Time remaining until end of journey(i.e. until max time)

Move: At each time step, salesman can take the following steps: - Up - Move one step up in the map - Down - Move one step down in the map - Right - Move one step right in the map - Left - Move one step left in the map.

Reward: Salesman/Agent gets a reward of -1 for each time step. If an order is delivered within that timestep, it gets a positive reward inversely proportional to the time taken to deliver. If all the orders are delivered and the agent is back to the restaurant, it gets an additional reward inversely proportional to time since start of episode.

AWS SageMaker: This toolkit allows training RL agents in cloud machines using docker containers.

Setup the environment: The environment implements the init(), step(), reset() and render() functions that describe how the environment behaves. This is consistent with Open AI Gym interfaces for defining an environment.

- Init() - initialize the environment in a pre-defined state
- Step() - take an action on the environment
- reset()- restart the environment on a new episode
- render() - get a rendered image of the environment in its current state.

```
# run in local mode?
local_mode=False

env_type="tsp-easy"

# create unique job name
job_name_prefix="rl-"+env_type

# S3 bucket
sage_session=sagemaker.session.Session()
s3_bucket=sage_session.default_bucket()
s3_output_path="s3://{}/".format(s3_bucket)
print("S3 bucket path: {}".format(s3_output_path))
```

Train the model : We make the problem much harder by randomizing the location of destinations each traversal. Hence, RL agent/salesman has to come up with a general strategy to navigate the grid. We will use the RL toolkit to train the agent.

```
%%time

iflocal_mode:
instance_type="local"
else:
instance_type="ml.m4.4xlarge"

estimator=RLEstimator(
entry_point="train-coach.py",
source_dir="src",
dependencies=["common/sagemaker_rl"],
toolkit=RLToolkit.COACH,
toolkit_version="1.0.0",
framework=RLFramework.TENSORFLOW,
role=role,
instance_type=instance_type,
instance_count=1,
output_path=s3_output_path,
base_job_name=job_name_prefix,
hyperparameters={
# expected run time 12 mins for TSP Easy
"RLCOACH_PRESET":"preset-"
+env_type,
},
)

estimator.fit(wait=local_mode)
```

7. Conclusion: In this paper we have tried to give an Reinforcement Learning environment for our training agent or the salesman to solve TSP problem more efficiently. GA is very much useful to solve local search problem of TSP where the dataset is small but if we get large data set due to over fitting and large optimization time the fitness of the optimizing path can be weak. In this case we can use the Reinforcement Learning to make the environment better to train the agent to choose random path from environment. No prior data set is needed to get optimal solution. This concept we can use to make other optimization like Ant-colony or Partial Swarm optimization to get better solution.

REFERENCES:

[1] L. Davis. “Job-shop Scheduling with Genetic Algorithms”. Proceedings of an International Conference on Genetic Algorithms and Their Applications, pp. 136-140, 1985. Zakir H. Ahmed International Journal of Biometrics & Bioinformatics (IJBB) Volume (3): Issue (6) 105

[2] I.M. Oliver, D. J. Smith and J.R.C. Holland. “A Study of Permutation Crossover Operators on the Travelling Salesman Problem”. In J.J. Grefenstette (ed.). Genetic Algorithms and Their Applications: Proceedings of the 2nd International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hilladale, NJ, 1987.

[3] http://en.wikipedia.org/wiki/Genetic_algorithm

[4] http://www.obitko.com/tutorials/genetic-algorithms/index.phphttp://www.talkorigins.org/faqs/genalg/genalg.html#examples:systems

[5] D.E.Goldberg, (1989) “Genetic Algorithms in search,Optimization and Machine Learning”, AddisonWesley Publishing Company, Ind. U.S.A.

[6] Melanie Mitche,(1998) ”An introduction to genetic Algorithm”, MIT press.

[7] NamitaKhurana, AnjuRathi,Akshatha P S,(2011) “Genetic Algorithm: A search of Complex, IJCA.
[8] http://en.wikipedia.org/wiki/Genetic_algorithm.

[9] Emanuel Falkenauer(1998) Genetic Algorithms and Grouping Problems. JohnWileyandSons.

[10] Kangshun Li,, Lanlan Kang, Wensheng Zhang, Bing Li, Comparative Analysis of Genetic Algorithm and Ant Colony Algorithm on Solving

[11] Traveling Salesman Problem, IEEE International Workshop on Semantic Computing and Systems

[12] "Yasuhiro TSUJIMURA and Mitsuo GEN”,(21-23 April 1998) Entropy-based Genetic Algorithm for Solving TSP, 1998 Second International

[13] Conference on Knowledge-Based Intelligent Electronic Systems, Adelaide, Australia. Editors, L.C. Jain and R.K.Jain

[14] “D. KAUR, M. M. MURUGAPPAN” Performance Enhancement in solving Traveling Salesman Problem using Hybrid Genetic Algorithm

[15] Vijendra Singh and SimranChoudhary (2009) , Genetic Algorithm for Traveling Salesman Problem: Using Modified Partially-Mapped Crossover

[16] Operator, IMPACT.

[17] Gen, 31. and R. Cheng(1997) Genetic Algorithms and Engineering Design. John Wiely& Sons.
[18] http://www.darwins-theory-of-evolution .com/

[19] http://www.obitko.com/tutorials/genetic-algorithms/encoding.ph

[20] Reinforcement learning for the traveling salesman problem with refueling
André L. C. Ottoni, Erivelton G. Nepomuceno, Marcos S. de Oliveira &Daniela C. R. de Oliveira