CAROMAD, Volume 02 (2017) : 8 – 21
http://www.laromad.usthb.dz

Cahiers de Recherche Opérationnelle
et Mathématiques de la Décision

The Traveling Salesman Problem: An Overview of Applications, Formulations, Exact and Approximate Algorithms

Ibtissam Ahmia & Asma Skoudarli

LaROMaD, Fac. Maths, USTHB, Pb 32, 16111 Bab Ezzouar, Algeria

iahmia@usthb.dz, a.skoudarli@hotmail.fr

Abstract: The traveling salesman problem (TSP) is a famous problem in the class of combinatorial optimization problems. Having a set of cities, the task of this problem is to find a route through these cities with shortest possible length. It is one of the most intensively studied problems in the field of operational research, this is due to the fact that it has practical and theoretical importance. It is easily formulated but yet difficult to be solved since it belongs to the NP-complete problems. It has been used and continues to be used as the benchmark problem for new algorithmic ideas. In this paper, we survey some formulations of the TSP, its applications and techniques that exist in the literature for solving it.

Key Words: Combinatorial Optimization,Traveling Salesman Problem, Operational Research

MSC(2010): 90C27, 05C38

Résumé : Le problème de voyageur de commerce (TSP) est un problème très connu de la classe des problèmes d'optimisation combinatoire. Etant donnée un ensemble de villes, l'objectif de ce problème est de trouver une tournée qui passe par toutes les villes. Il est l'un des problèmes les plus étudiés de recherce opérationnelle et ceci est dÃ au fait qu'il a une importance pratique et théorique. Il est facilement formulé mais difficile Ã résoudre puisqu'il appartient Ã la classe NP-complet. Il a été utilisé et continue Ã Ã ªtre utilisé pour tester l'efficacité de nouveaux algorithmes. Dans cet article, une synthèse est présentée regroupant: Les différentes formulations de ce problème, ses applications, et les techniques qui existent dans la littérature pour le résoudre.

Mots clés : Optimisation Combinatoire, Le problème de voyageur de commerce, Recherche Opérationnelle

1 Introduction

The Traveling Salesman Problem (TSP) is a classical combinatorial optimization problem, which is simple to state but very difficult to solve. The problem is to find the shortest tour through a set of N cities so that each city is visited exactly once. Formally, the TSP can be stated as follows. The distances between N cities are stored in a distance matrix D with elements d_{ij} where i, j ={ 1,. . . ,n} and the diagonal elements d_{ij} are zero. A tour can be represented by a cyclic permutation π of $\{1, 2, \ldots, n\}$ where π_i represents the city that follows city i on the tour. The traveling salesman problem is then the optimization problem to find a permutation π that minimizes the length of the tour. This problem is known to be NP-complete, which means that we don't know yet an algorithm that can solve it exactly in polynomial time. However, researchers keep developing and ameliorating algorithms for solving TSP since it is an important part of applications in many areas including vehicle routing, computer wiring, machine sequencing, crystallography, and many other applications. In this paper we give a very brief overview of the TSP, it is organized as follows: In section 2 different formulations of TSP are presented. In section 3, we present some applications of this problem. Section 4 provides the different approaches proposed to solve it.

2 The Traveling Salesman problem (TSP) formulations

In this section, we briefly summarize some formulations that exist in the literature for the TSP.

2.1 Graph formulation

The traveling salesman problem has been firstly lunched based on the graph theory in 1930's [21]. Let $G = (V, E)$ be a complete graph where the cities correspond to the node set $V = \{1, 2, \ldots, n\}$, $E = \{(r, s) : r, s \in V\}$ the edge set and each edge $e_i \in E$ has an associated weight d_{ij} representing the distance between the nodes it connects. If the graph is not complete, the missing edges can be replaced by edges with very large distances. The TSP consists of determining a minimum distance circuit passing through each node once and only once. Such a circuit is known as a tour or Hamiltonian circuit (or cycle). If $d_{ij} = d_{ji}$ for any edge in the graph, we say that we have a symmetric TSP denoted (sTSP) otherwise it means that the distances are not equal for all pairs of cities the TSP is called asymmetric and is denoted (aTSP).

2.2 Integer programming formulations

Many mathematical formulations of this problem exist in the literature we have selected few of them from the survey of Orman, A. J. and Williams, H. Paul. "A Survey of Different Integer Programming Formulations of the Traveling Salesman Problem" [16].

In all following formulations the set of cities is $N = \{1, 2, ..., n\}$ and variables are defined as:

$$x_{ij} = \begin{cases} 1 & \text{iff edge(i,j) is a link in the tour} \\ 0 & \text{otherwise} \end{cases}$$

The objective function is:

$$\text{Minimize} \sum_{\substack{i,j \\ i \neq j}} c_{ij} x_{ij}$$

where c_{ij} is the length of edge (i, j).

2.3 Formulation of Danzing, Fulkerson and Johnson (1954)

$$\begin{cases} \sum\limits_{\substack{j \\ i \neq j}} x_{ij} = 1 & \forall i \in N \quad (1) \\ \sum\limits_{\substack{i \\ i \neq j}} x_{ij} = 1 & \forall j \in N \quad (2) \\ \sum\limits_{\substack{i,j \in M \\ i \neq j}} x_{ij} \leq \mid M \mid -1 & \forall M \subset N \text{ such that } \{1\} \notin M, \mid M \mid \geq 2 \quad (3) \end{cases}$$

Constraints (1) and (2), define a regular assignment problem, where (1) ensures that each city is entered from only one other city, (2) ensures that each city is only departed to one other city and constraint (3) eliminates subtours.
The numbers of constraints and variables in this formulation are respectively:
$2^{n-1} + n - 1$ and $n(n-1)$.
A variant of this formulation results from replacing the third constraint by:

$$\sum_{\substack{i \in M \\ j \in \overline{M}}} x_{ij} \geq 1 \qquad \forall M \subset N \text{ where } \{1\} \notin M \text{ and } \overline{M} = N - M \quad (4)$$

2.4 Formulation of Miller, Tucker and Zemlin (1960)

In addition of constraint 1 and 2 a new constraint is formulated using continuous variables defined as follows:
u_i= sequence in which city i is visited $(i \neq 1)$

The new constraint is:

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad \forall i, j \in N - 1, i \neq j \qquad (5)$$

This formulation has $n^2 - n + 2$ constraints, $n(n-1)$ 0-1 variables and $(n-1)$ continuous variables.

2.5 Formulation of Finke, Claus and Gunn (1983)

The assignment constraints (1) and (2) are retained and a new constraints are added which where formulated using continuous variables:
y_{ij}= 'Flow' of commodity 1 in arc (i, j), $i \neq j$.
z_{ij}= 'Flow' of commodity 2 in arc (i, j), $i \neq j$.
Constraints are:

$$\begin{cases} \sum_{\substack{j \\ j\neq 1}} (y_{1j} - y_{j1}) = n - 1 & & (6) \\ \sum_{j} (y_{ij} - y_{ji}) = 1 & \forall i \in N - \{1\}, i \neq j & (7) \\ \sum_{\substack{j \\ j\neq 1}} (z_{1j} - z_{j1}) = -(n-1) & & (8) \\ \sum_{j} (z_{ij} - z_{ji}) = -1 & \forall i \in N - \{1\}, i \neq j & (9) \\ \sum_{j} (y_{ij} + z_{ij}) = n - 1 & & (10) \\ y_{ij} + z_{ij} = (n-1)x_{ij} & \forall i, j \in N & (11) \end{cases}$$

This formulation has $n(n+4)$ constraints, $n(n-1)$ 0-1 variables and $2n(n-1)$ continuous variables.

3 Applications [9], [14]

Applications of the TSP and its variations spans over several areas of knowledge including mathematics, computer science, operations research, genetics,engineering, and electronics. In this section, some selected applications of TSP are presented.

3.1 Computer wiring (Lenstra and Rinnooy Kan,1975)[11]

Some computer systems can be described as modules with pins attached to them. It is often desired to link these pins by means of wires, so that exactly two wires are attached to each pin and total wire length is minimized.

3.2 Hole punching (Reinelt, 1989)[20]

In several manufacturing contexts, it is necessary to punch holes on boards or metallic sheets. The problem consists of determining a minimum-time punching sequence. Such a problem occurs frequently in metallic sheet manufacturing and in the construction of circuit boards. These problems are often of large scale and must be solved in real time.

3.3 Job sequencing

Suppose n jobs must be performed sequentially on a single machine and that c_{ij} is the change-over time if job j is executed immediately after job i. Then again, by introducting a dummy job, this problem can be formulated as a TSP.

3.4 Crystallography (Bland and Shallcross,1989) [1]

In crystallography, some experiments consist of taking a large number of X-ray intensity measurements on crystals by means of a detector. Each measurement requires that a sample of the crystal be mounted on an apparatus and that the detector be positioned appropriately. The

order in which the various measurements on a given crystal are made can be seen as the solution of a TSP. In practice, these problems are of large scale and obtaining good TSP solutions can reduce considerably the time needed to carry out all measurements.

3.5 The order-picking problem in warehouses

This problem is associated with material handling in a warehouse (Ratliff and Rosenthal,1983). Assume that at a warehouse an order arrives for a certain subset of the items stored in the warehouse. Some vehicle has to collect all items of this order to ship them to the customer. The relation to the TSP is immediately seen. The storage locations of the items correspond to the nodes of the graph. The distance between two nodes is given by the time needed to move the vehicle from one location to the other. The problem of finding a shortest route for the vehicle with minimum pickup time can now be solved as a TSP.

3.6 Overhauling Gas Turbine Engines

This application was reported by Plante, Lowe and Chandrasekaran (1987)[19] and occurs when gas turbine engines of aircrafts have to be overhauled. To guarantee a uniform gas flow through the turbines there are so-called nozzle-guide vane assemblies located at each turbine stage. Such an assembly basically consists of a number of nozzle guide vanes affixed about its circumference. All these vanes have individual characteristics and the correct placement of the vanes can result in substantial benefits (reducing vibration, increasing uniformity of flow, reducing fuel consumption). The problem of placing the vanes in the best possible way can be modeled as a symmetric TSP.

3.7 Vehicle routing

Suppose that in a city n mail boxes have to be emptied every day within a certain period of time, say one hour. The problem is to find the minimum number of trucks to do this and the shortest time to do the collections using this number of trucks. As another example, suppose that n customers require certain amounts of some commodities and a supplier has to satisfy all demands with a fleet of trucks. The problem is to find an assignment of customers to the trucks and a delivery schedule for each truck so that the capacity of each truck is not exceeded and the total travel distance is minimized. Several variations of these two problems, where time and capacity constraints are combined, are common in many real world applications. This problem is solvable as a TSP if there are no time and capacity constraints and if the number of trucks is fixed (say m). In this case we obtain an m salesmen problem. Nevertheless, one may apply methods for the TSP to find good feasible solutions for this problem (see Lenstra and Rinnooy Kan, 1974 [10]) .

4 Approaches to solve the Traveling Salesman Problem

Many approaches proposed to solve the TSP and its variants exist in the literature. In this section we present an overview of this approaches that are classified as follows: Exact methods, heuristics and meta-heuristics.

4.1 Exact methods

A large number of exact algorithms have been proposed for the TSP. These methods provide an optimal tour for TSP. They have proved their effectiveness for small instances of the TSP but since they need a high computational time, they are not preferred for practical problems instances where it is important to get a fast solution.

4.1.1 Brute force method

The brute-force method generate all possible tours and compute their distances. The shortest tour is thus the optimal tour. To solve TSP using Brute-force method we can use the following steps:

- **Step 1**: Enumerate the total number of hamiltonian tours.
- **Step 2**: Draw and list all the possible hamiltonian tours.
- **Step 3**: Calculate the distance of each tour.
- **Step 4**: Choose the shortest tour; this is the optimal solution.

For n cities in a graph there are $(n-1)!$ maximum possible hamiltonian tours for aTSP and $(n-1)!/2$ maximum possible hamiltonian tours for sTSP. Therefore, this algorithm which will not only find these tours but also compare them is certainly going to be $O(n!)$ for aTSP and $O(n!/2)$, which is extremely inefficient.

4.1.2 Branch and Bound

Branch and bound algorithms are commonly used to solve the TSP. Firstly Dantzig et al. [2] applied the method to the aTSP. A more general description was provided by Land and Doig [8] in the context of solving integer programming problems by linear programming. Finally, the approach was described and named branch and bound by Little et al. [13] in an application to the TSP.
The branch and bound strategy divides a problem to be solved into a number of sub-problems. It is a system for solving a sequence of sub-problems each of which may have multiple possible solutions and where the solution chosen for one sub-problem may affect the possible solutions of later sub-problems. This method are based on some relaxation of the integer linear programming model of TSP.
Formulation of TSP by using branch and bound technique is given by these steps:

- **Step 1**: Choose a start node.
- **Step 2**: Set bound to a very large value, let's say infinity.
- **Step 3**: Choose the cheapest arc between the current and unvisited node and add the distance to the current distance and repeat while the current distance is less than the bound.
- **Step 4**: If current distance is less than bound, then we are done.

- **Step 5**: Add up the distance and bound will be equal to the current distance.
- **Step 6**: Repeat step 5 until all the arcs have been covered.

The branch and bound technique was able to increase the size of the problem solvable without using any problem specific methods. The algorithm branches into the set of all possible tours while calculating the lower bound on the length of the tour for each subset. Eventually it finds a single tour in a subset whose length is less than or equal to some lower bound for every tour. The algorithm however grows exponentially in time with the input, but it is able to calculate the TSP for 40 cities with appreciable average time consumption as displayed by the authors.

4.1.3 Cutting plane

The cutting plane method was introduced by Dantzig, Fulkerson, and Johnson [2] on the traveling salesman problem. It can be used to attack any problem:

$$\begin{cases} \text{minimize} & c^T x \\ \text{s.t.} & x \in S \end{cases} \tag{1}$$

Where S is a finite subset of some Euclidean space $\mathbb{R}^m$, provided that an efficient algorithm to recognize points of S is available. This method is iterative; each of its iterations begins with a linear programming relaxation of (1), meaning a problem

$$\begin{cases} \text{minimize} & c^T x \\ \text{s.t.} & Ax \leq b \end{cases} \tag{2}$$

Where the polyhedron P defined as $\{x : Ax \leq b\}$ contains S and is bounded. Since P is bounded, we can find an optimal solution x^* of (2) which is an extreme point of P. If x^* belongs to S, then it constitutes an optimal solution of (1); otherwise, some linear inequality separates x^* from S in the sense of being satisfied by all the points in S and violated by x^*; such an inequality is called a cutting plane or simply a cut.

4.1.4 Branch and Cut method

This method is a combination of a branch-and-bound algorithm and a cutting plane method. The first version of this idea was applied to the TSP by Hong (1972)[5] and Miliotis (1976)[15]. Next, Padberg and Rinaldi(1987, 1991)[17, 18] used it to solve very large TSP instances (up to 2000 cities).

The branch and cut method consists of solving the linear program without the integer constraint using the simplex algorithm. When an optimal solution is obtained, and this solution has a non-integer value for a variable that is supposed to be integer, a cutting plane algorithm is used to find additional linear constraints which are satisfied by all feasible integer points but violated by the current fractional solution. If such an inequality is found, it is added to the formulation, such that resolving it will yield a different solution which is less fractional. This process is repeated until either an integer solution is found (which is then known to be optimal) or until no more cutting planes are found. If we have not finished with an optimal solution to the integer problem, we applied branch and bound algorithm.

This method has been successful in finding optimal solutions of large instances of sTSP. However, compare to TSP, the amount of research carries out on branch and cut applied to CVRP is still

quite limited. Similar to in branch and bound algorithms, the central problem of branch and cut is that the tree generated by the branching procedure becomes too large and termination seems unlikely within a reasonable amount of time.

4.2 Approximate methods

The approximate approaches need a short time of execution which makes them more useful for the applications that prefer computational time of the algorithm over the accuracy of its result. A vast research studies exist in literature to solve the TSP approximately. The most relevant and mostly used approaches are listed in this section.

4.2.1 Tour construction approaches

Greedy Approach
The greedy heuristic constructs a tour iteratively, by selecting the shortest edge and adding it to the tour as long as it doesn't create a cycle with less than n edges, or increases the degree of any node to more than 2. We must not add the same edge twice of course.
Steps of Greedy approach are given in Algorithm 1.

Algorithm 1: TSP using Greedy Approach.

- **Step 1**: Sort all edges.
- **Step 2**: Select the shortest edge and add it to our tour if it doesn't violate any constraints of TSP.
- **Step 3**: If we have n edges in our tour than we have an optimal solution; write down the tour of the optimal solution and calculate their distance. If no, repeat step 2.

The Greedy algorithm normally keeps solution within $15 - 20\%$ of the Held-Karp lower bound [6]. Complexity of the greedy heuristic is $O(n^2 log_2(n))$.

Nearest neighbor heuristic
The key of this approach is to always visit the closest city. It selects a starting node and then always selects the nearest city to be added to the tour, it then walks to that city and repeats by choosing a new non-selected city, until all cities is in the tour. To complete the tour, an edge is added between the last selected city and the starting city.
Algorithm 2 shows the methodology to solve TSP by Nearest Neighbor Heuristic.

Algorithm 2: TSP using Nearest Neighbor Heuristic.

- **Step 1**: Select a random city to be a starting node.
- **Step 2**: Look at all the arcs coming out of the starting node that have not been visited and choose the next nearest node.
- **Step 3**: Repeat the process until all the nodes have been visited at least once.
- **Step 4**: Check if all nodes are visited. If so return to the first point which gives us a tour.

- **Step 5**: Draw and write down the tour, and calculate the distance of the tour.

A general version of this heuristic has running time of $O(n^2)$.
A common way of measuring the performance of TSP heuristics is to compare its results to the Held- Karp (HK) lower bound. The Nearest Neighbor heuristic approach generally keeps its tour within 25% of the Held-Karp lower bound[6].

Insertion heuristic
There are many variants of this approach. All insertion algorithms start with a tour consisting of an arbitrary city and then choose in each step a city not yet on the tour. This city is inserted into the existing tour between two consecutive cities, such that the insertion cost (i.e., the increase in the tour's length) is minimized. The algorithms stop when all cities are on the tour.

Algorithm 3: TSP using Insertion heuristic.

- **Step 1**: Select the shortest edge, and make a subtour of it.
- **Step 2**: Select a city not in the subtour, having the shortest distance to any one of the cities in the subtour.
- **Step 3**: Find an edge in the subtour such that the cost of inserting the selected city between the edge's cities will be minimal.
- **Step 4**: Repeat step 2 until no more cities remain.

The complexity with this type of heuristic approach is given as $O(n^2)$.

Christofide heuristic
This heuristic is named after NicosChristofides. The goal of this algorithm is to find a solution to the instances of the TSP where the edge weights satisfy the triangle inequality. The steps of this heuristics are gievn below in algorithm 4.

Algorithm 4: TSP using Christofide heuristic.

- **Step 1**: Construct a minimal spanning tree (MST) from the set of all cities.
- **Step 2**: Create a minimum-weight matching (MWM) on the set of nodes having with odd degree. Combine the MST with the MWM.
- **Step 3**: Create an Euler cycle from the combined graph, and traverse it taking shortcuts to avoid visited nodes.

Tests have shown that Christofides' algorithm tends to place itself around 10% above the Held-Karp lower bound [6]. Complexity of this approach is $O(n^3)$.

More information on tour construction heuristics can be found in (Johnson & McGeoch, 2002) [7].

4.2.2 Tour improvement

Once a tour has been generated using any tour construction heuristic, we can improve that solution by applying an improvement heuristic. There are several ways to do this, but the most popular ones are the 2-opt and 3-opt exchange heuristic. Their performances are depend on the tour generated by the tour construction heuristic. Other ways of improving the solution is to apply meta-heuristic approaches such as tabu search, simulated annealing or genetic algorithm using 2-opt and 3-opt.

2-opt and 3-opt
The 2-opt algorithm removes randomly two edges from the tour, and reconnects the new two paths created. This is often refered as a 2-opt move. There is only one way to reconnect the two paths so that we still have a valid tour (Figure.1). This is done only if the new tour is shorter than older. Continue removing and reconnecting the tour until no 2-opt improvements can be found. The resulting tour is now 2-optimal.
For n cities in the TSP, the 2-Opt algorithm consists of the steps shown in Algorithm 5.

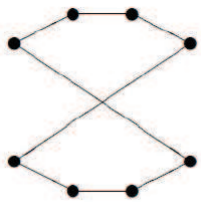

Figure 1: 2-opt move

Algorithm 5: TSP using 2-Opt heuristic.

- **Step 1**: Let S be the initial solution provided by the user and z its objective function value. Set $S^* = s$, $z^* = z$, $i = 1$ and $j = i + 1 = 2$.
- **Step 2**: Consider the exchange results in a solution S that has objective function value $z' < z^*$, set $z^* = z'$ and $S^* = S'$. If $j < n$ repeat Step 2; otherwise set $i = i + 1$ and $j = i + 1$. If $i < n$, repeat Step 2; otherwise go to Step 3.
- **Step 3**: If $S \neq S^*$, set $S = S^*$, $z = z^*$, $i = 1$, $j = i + 1 = 2$ and go to Step 2. Otherwise, output S^* as the best solution and terminate the process.

The complexity of the 2-opt search is $O(n^2)$.

The 3-opt algorithm works in a similar fashion, but instead of removing two edges we remove three. This means that we have two ways of reconnecting the three paths into a valid tour (Figure.2). Search is finished when no more 3-opt moves can improve the tour. If a tour is 3 optimal it is also 2 optimal [4].

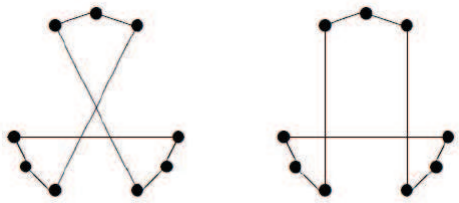

Figure 2: 3-opt move

Running the 2-opt move often results in a tour with a length less than 5% above the Held-Karp bound. The improvements of a 3-opt move usually generates a tour about 3% above the Held-Karp bound [6].

k-opt

2-opt and 3-opt are special cases of k-opt exchange heuristic. We don't necessarilly have to stop at 3-opt, we can continue with 4-opt and so on, but exchange heuristic having $k > 3$ will take more computational time. Mainly one 4-opt move is used, called "the crossing bridges" (Figure.3). This particular move can not be sequentially constructed using 2-opt moves. For this to be possible two of these moves would have to be illegal [4].

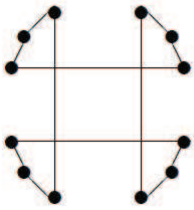

Figure 3: Double bridge move

Lin-Kernighan

The Lin-Kernighan heuristic [12] is generally considered to be one of the most effective methods for generating optimal or near-optimal solutions for the sTSP.
The LK heuristic is a variable k-way exchange heuristic. It decides which k is the most suitable at each iteration step. This makes the algorithm quite complex, and few have been able to make improvements to it.
The time complexity of LK is approximately $O(n^{2.2})$ [4], making it slower than a simple 2-opt implementation. Lin & Kernighan constructed an algorithm making it possible to get within 2% of the Held- Karp lower bound.

Tabu Search (TS)

It is a neighborhood-search algorithm which seacrh among the neighbors of a candidate to find a better one. In general, when using a neighborhood-search on the TSP, neighboring moves are often normal 2-opt exchange. A problem with simple neighborhood search approach is that one can easily get stuck in a local optimum. This can be avoided easily in TS approach.
The TS will allow moves with negative gain if we can not find a positive one. By allowing negative gain me may end up running in circles, as one move may counteract the previous.
To avoid this TS keeps a tabu list containing bad moves. After moving to a neighboring solution

the move will be put on the tabu-list and will thus never be applied again unless it improves our best tour or the tabu has been pruned from our list.
There are several ways of implementing the tabu list. One involves adding the two edges being removed by a 2-opt move to the list. A move will then be considered tabu if it tries to add the same pair of edges again. Another way is to add the shortest edge removed by a 2-opt move, and then making any move involving this edge tabu. Other methods keep the endpoints of each move, making a move tabu if it uses these endpoints. For more detail paper [6] can be referred.

The big problem with the TS is its running time. Most implementations for the TSP generally takes $O(n^3)$ [6], making it far slower than a 2-opt local search.

Simulated annealing (SA)
Simulated Annealing (SA) has been successfully adapted to give approximate solutions for the TSP. SA is basically a randomized local search algorithm allowing moves with negative gain.
A baseline implementation of SA for the TSP is presented in [6]. They use 2-opt moves to find neighboring solutions.
In SA, Better results can be obtained by increasing the running time of the SA algorithm, and it is found that the results are comparable to the LK algorithm.
Due to the 2-opt neighborhood, this particular implementation takes $O(n^2)$ with a large constant of proportionality [6].

Genetic algorithm
Genetic Algorithm (GA) is based on genetics. A basic GA starts with a randomly generated population of candidate solutions. Some (or all) candidates are then mated to produce offspring and some go through a mutating process. Each candidate has a fitness value telling us how good they are. By selecting the most fit candidates for mating and mutation the overall fitness of the population will increase. Applying GA to the TSP involves implementing crossover routine, measure of fitness, and also mutation routine. A good measure of fitness is the actual length of the candidate solution. Different approaches of crossover and mutation routines are discussed in [6].

Algorithm 5: TSP using Genetic Algorithm.

- **Step 1**: Obtain the maximum number of individuals in the population P and the maximum number of generations G from the user, generate P solutions for the first generation's population randomly, and represent each solution as a string. Set generation counter $Ngen = 1$.
- **Step 2**: Determine the fitness of each solution in the current generation's population and record the string that has the best fitness.
- **Step 3**: Generate solutions for the next generation's population as follows:
  1. Retain $0.1P$ of the solutions with the best fitness in the previous population.
  2. Generate $0.89P$ solutions via mating, and
  3. Select $0.01P$ solutions from the previous population randomly and mutate them.
- **Step 4**: Update $Ngen = Ngen + 1$. If $Ngen \leq G$, go to Step 2. Otherwise stop.

Ant Colony Optimization (ACO)
Ant Colony Optimization is a meta-heuristic technique that is inspired by the behavior of real

ants. Real ants cooperate to find food resources by laying a trail of a chemical substance called "pheromone" along the path from the nest to the food source. Depending on the amount of pheromone available on a path, new ants are encouraged, with a high probability, to follow the same path, resulting in even more pheromone being placed on this path. Shorter routes to food sources have higher amounts of pheromone. Thus, over time, the majority of ants are directed to use the shortest path. This type of indirect communication is called "stigmergy", in which the concept of positive feedback is exploited to find the best possible path, based on the experience of previous ants. For more detailed information on ant colony optimization for the TSP see [3].

5 Conclusion

The traveling salesman problem (TSP) occupies very important place in Operational Research. This survey is limited to some selected formulations, applications and resolution methods of this problem because it is not possible to cover all its applications and approaches in one article. For deep study of the TSP problem we recommend the book "The Traveling Salesman Problem: A guided tour of combinatorial optimization" edited by Lawler, Lenstra, Rinooy Kan and Shmoys [22] which provides the state of the art description of the TSP, and "The Traveling Salesman Problem and Its Variations" edited by Gregory Gutin and Abraham P.Punnen [23].

References

[1] R. G. Bland, and D . F. Shallcross. *Large traveling salesman problems arising experiments in X-ray crystallography: A preliminary report on computation*,Operations Research Letters 8, 125-128, 1989.

[2] G. Dantzig, R. Fulkerson, and S. Johnson. *Solution of a large scale traveling salesman problem.* Operations Research, 2:393–410, 1954.

[3] M. Dorigo, L.M. Gambardella. *Ant Colonies for the Traveling Salesman Problem.* University Libre de Bruxelles, Belgium, 1996.

[4] K. Helsgaun. *An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic.* Department of Computer Science, Roskilde University.

[5] S. Hong, *A Linear Programming Approach for the Traveling Salesman Problem.* Ph.D. Thesis, The Johns Hopkins University, 1972.

[6] D.S. Johnson, and L.A. McGeoch. *The Traveling Salesman Problem: A Case Study in Local Optimization.* November 20, 1995.

[7] D.S. Johnson, and L.A. McGeoch. *Experimental Analysis of Heuristics for the STSP, The Traveling Salesman Problem and its Variations.* Gutin & Punnen (eds), Kluwer Academic Publishers, pp. 369-443, 2002.

[8] A. H. Land and A. Doig. *An automatic method for solving discrete programming problems.* Econometrica, 28:497–520, 1960.

[9] G. Laporte. *The vehicle routing problem: An overview of exact and approximate algorithms.* European journal of operational research, 59 (3), 345-358.

[10] J. K. Lenstra, A. H. G. Rinnooy Kan *Some Simple Applications of the Traveling Salesman Problem*Research Report, Stichting Mathematisch Centrum, Amsterdam, 1974.

[11] J. K. Lenstra, A. H. G. Rinnooy Kan. *Some simple applications of the traveling salesman problem.*Operational Research Quarterly 26, 717-733, 1975.

[12] S. Lin and B.W. Kernighan. *An effective heuristic algorithm for the traveling salesman problem.* Operations Research 21 (1973) ,498-516.

[13] J. D. Little, K. G. Murty, D. W. Sweeney, and C. Karel. *An algorithm for the traveling salesman problem.* Operations Research, 11(6):972–989, 1963.

[14] R. Matai, S. Singh, ML. Mittal. *Traveling salesman problem: an overview of applications, formulations, and solution approaches.* In: Traveling salesman problem, theory and applications. Intech (2010).

[15] P. Miliotis. *Integer programming approaches to the traveling salesman problem.* Mathematical Programming 10, 367–378, 1976.

[16] A. J. Orman,and H. Paul Williams. *A survey of different integer programming formulations of the traveling salesman problem.* Operational Research working papers, LSEOR 04.67. Department of Operational Research, London School of Economics and Political Science, London, UK, 2004.

[17] M. Padberg and G. Rinaldi, *Optimization of a 532-city symmetric traveling salesman problem by branch and cut.* Operations Research Letters 6, 1–7, 1987.

[18] M. Padberg and G. Rinaldi. *branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems.* SIAM Review 33, 60–100, 1991.

[19] R. D. Plante, T. J. Lowe and R. Chandrasekaran. *The Product Matrix Traveling Salesman Problem: An Application and Solution Heuristics.* Operations Research, 35, 772-783, 1987.

[20] G. Reinelt. *Fast heuristics for large geometric traveling salesman problems,*Report No. 185, Institut of Mathematics, University of Augsburg, 1989.

[21] M. Bellmore and G.L. Nemhauser. *The Traveling Salesman Problem: A Survey* . Operation Research, 16: 538–558, 1986.

[22] E.L. Lawler,J.K. Lenstra, A.H.G. Rinnooy Kan,D.B. Shmoys. *The Traveling Salesman Problem: A guided tour of combinatorial optimization.* John Wiley and Sons,1985

[23] G. Gutin and A.P. Punnen. *The Traveling Salesman Problem and Its Variations.* Springer US,2007