process called heuristics that provides probability outcomes for this type of equation.

In the early days of computers, the traveling salesman problem was one example of the many tasks that computers could do more efficiently than humans. A simple computer program written in almost any programming language can provide excellent and actual results for solving the traveling salesman problem with any reasonable amount of complexity [3].

In modern IT, the equation itself has applications in identifying network or hardware optimization methods. For example, in the vastly complex global Internet, the traveling salesman problem can be used to work out the most efficient trajectories for data packets being routed anywhere in the system. The same holds true for private networks. Because of the hardness of NP-complete problems, no one has found a polynomial-time algorithm for TSP. Therefore, much research has concentrated on approximation algorithms whose goal is to find near optimal rather than optimal tours, but it does not mean that it is impossible to solve any large instances of such problems [10].

Traveling salesman has to visit all of his customers, come back home and take the shortest way. The way in which the visits to all points in the graph can be modeled is implemented with Hamilton's sequence. The sequence in a graph G is Hamilton's sequence in graph G which contains all the points of graph G. This is even the definition of Hamilton's way and Hamilton cycle, because both are special cases of Hamilton's sequence. Graph G is Hamilton's graph, if in graph G exists Hamilton's cycle. The target is in coherent edge rated graph to find the shortest Hamilton's cycle / the shortest closed Hamilton's sequence [10].

The optimization version of traveling salesman problem belongs to NP-hard (NP means non-polynomial) problems. In general case it is not known how to find an optimal solution for every possible input in real time nor is it yet known. If a polynomial complex algorithm can exist [10].

Algorithm is a mechanical procedure containing any simple element steps which for any input produces an output. Algorithm can be assigned for:

• verbal description in general language

• as computer program in any programming language

• as hardware circuit etc.

Time of compute means that computers compute really fast (billions of operations in second) but of course not infinitely fast. To run any operation takes some non-null time. To compute for the same input can cost different time-consumption for different algorithms.

This could depend on:

- implementation of algorithm
- programming language
- used interpreter
- used hardware (processor, memory etc.).

Polynomial algorithms, class P and NP can be defined as follows: In theory of complexity there is class P one of the basics classes. It contains all problems which are resolvable with deterministic Turing machine in polynomial time. P is obviously considered as class of problems which can be effectively resolvable but even in this class it is possible to find effectively non-resolvable problems (with complexity e.g. N1000000 etc.).

P contains many natural problems, looking for the smallest common divisor or finding the maximal paring in graph. In 2002 was proved that the problem decision of primarily belongs to class P. Superset P is NP. Class NP is a class of problems resolvable in polynomial time consuming on non-deterministic Turing machine.

The relationship between P and NP is not still resolved. But here is possibility that these two sets are equal. Even if the proof still does not exist, most of the word wide experts think that P is a proper subset NP. In real world these kinds of problems are solved just approximately – with the use of heuristic algorithm (genetic algorithm, tabu search etc.). With these heuristic algorithms relatively good solutions can be achieved but the cost of this is to give up the entitlement for optimal solution [10].

It should not be forgotten that, even a heuristic algorithm can find an optimal solution. Traveling salesman problem can be split into two different kinds of problems – asymmetric and symmetric. In the symmetric case of the traveling salesman problem is the distance between two points which are the same in both directions.

The basic structure of graph is an undirected graph. In asymmetric case the distance between two points in both directions can be different or there can be just a way in one direction from/to the point. This structure is called oriented graph. In the example for asymmetric traveling salesman problem it can be looking for the shortest tour in a city with one-way streets. The most used case type of traveling salesman problem is so-called metric traveling salesman. In this case all distances are in graph fulfills triangular inequality. This type of traveling salesman problem coincides with the most of real problems. This attitude allows construction of approximate algorithms [10].

The algorithms for solving TSP can be divided into four classes: exact algorithms, heuristic algorithms, approximate algorithms and metaheuristics algorithms. The exact algorithms are guaranteed to find the optimal solution. However, these algorithms are quite complex and very demanding of computer power. The heuristic algorithms can obtain good solutions but it cannot be guarantee that the optimal solution will be found [10].

In general, the heuristic algorithms are subdivided into the following three classes: tour construction algorithms, tour improvement algorithms and hybrid algorithms. The tour construction algorithms gradually build a tour by adding a new city at each step, the tour improvement algorithms improve upon a tour by performing various exchanges, and finally hybrid algorithms use both composing and improving heuristics at the same time. The best results are obtained by using hybrid approaches [10].

The Travelling Salesman Problem is a traditional algorithm used to find the shortest or least path This number of routes can cause congestion, delay or data loss, so utilizing Travelling Salesman Problem to find the shortest path from sender to receiver is an efficient way to transmit data with less congestion and also in less time. It is generally used to find the optimum solution for a complex network [10].

The Travelling Salesman Problem can be solved using meta-heuristic approaches such as Ant colony optimization or genetic algorithm, however the easiest approach to solve the Travelling Salesman Problem is Nearest Neighbor Algorithm. The optimum solution of Travelling Salesman will save time and travelling expenses of a salesman. In Travelling Salesman problem, the main aim is to visit each city in a network once following the shortest path and return to the starting position [10].

2.1 Exhaustive Search

In computer science, brute-force search or exhaustive search, also known as generate and test, is a very general problem-solving technique and algorithmic paradigm that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement [8].

A brute-force algorithm to find the divisors of a natural number *n* would enumerate all integers from 1 to n, and check whether each of them divides *n* without remainder. A brute-force approach for the eight queens puzzle would examine all possible arrangements of 8 pieces on the 64-square chessboard, and, for each arrangement, check whether each (queen) piece can attack any other [8].

While a brute-force search is simple to implement, and will always find a solution if it exists, its cost is proportional to the number of candidate solutions – which in many practical problems tends to grow very quickly as the size of the problem increases (combinatorial explosion). Therefore, brute-force search is typically used when the problem size is limited, or when there are problem-specific heuristics that can be used to reduce the set of candidate solutions to a manageable size. The method is also used when the simplicity of implementation is more important than speed [8].

The simplest attitude to solve the problem of travelling salesman is to try all the 12 possible routes/all possible combinations of cities to visit while summing distances between the cities in particular route and finally find the route with shortest total distance. This very straight-forward solution is called brute-force or exhaustive search [8].

However, this technique has a shortcoming of immense importance – running time of $\Theta((n\text{-}1)!)$, i.e. it needs to generate $(n\text{-}1)!$ permutations of *n* cities and calculate the total distance of it. And as *n* grows, the factorial $(n\text{-}1)!$ becomes larger than all polynomials and exponential functions (but slower than double exponential functions) in n-1. This enormous growing of possible routes means enormous growing of time needed for solving TSP even for contemporary computers [8].

On the other side, there is no doubt the result will be surely the shortest route of all existing. Unfortunately, due to its time complexity $\Theta(N!)$ this algorithm is not very suitable for the purpose of this work and so if there exists other algorithm with better time complexity but still exact (thus returning the route of the same quality as brute-force algorithm, i.e. optimal route), it should be chosen for the implementation rather than brute force [8].

This is the case, for example, in critical applications where any errors in the algorithm would have very serious consequences; or when using a computer to prove a mathematical theorem. Brute-force search is also useful as a baseline method when benchmarking other algorithms or metaheuristics. Indeed, brute-force search can be viewed as the simplest metaheuristic. Brute force search should not be confused with backtracking, where large sets of solutions can be discarded without being explicitly enumerated (as in the textbook computer solution to the eight queens problem above). The brute-force method for finding an item in a table – namely, check all entries of the latter, sequentially – is called linear search [8].

2.2 Travelling Salesman Problem

The traveling salesman problem consists of a salesman and a set of cities. The salesman has to visit each one of the cities starting from a certain one (e.g. the hometown) and returning to the same city. The challenge of the problem is that the traveling salesman wants to minimize the total length of the trip.

A. Formulation

In order to formalize the definition of the traveling salesman problem, several basic computer scientific terms must be defined. The first is that of a graph. This term is used to describe a set of vertices, or nodes, and a set of edges that connect the vertices. A complete graph has an edge between all pairs of vertices. These edges can be directed or undirected and can have weights associated with them.

A path between two vertices is a sequence of edges that begins at one vertex and ends at another vertex. A cycle is a path that begins and ends at the same vertex. A Hamiltonian cycle is a cycle covering all nodes in the graph exactly once [3]. The Traveling Salesman Problem can now be formally defined as follows:

Determine the shortest Hamiltonian Cycle in a complete weighted graph.

B. Solution Algorithms

There are three types of approach or solving NP complete problems:

• Devising algorithms for finding exact solution (they will work reasonably fast only for relatively small problem sizes)

•Devising “sub-optimal” or heuristic algorithms that deliver seemingly or provably good solutions, but which could not prove to be optimal.

• Finding special cases for the problem for which either exact or better heuristic are possible.

In particular, here are the Solution algorithms for TSP:

2.2.1. Exact Algorithms

The exact algorithms are guaranteed to find an optimal solution but may take an exponential number of iterations. This means that you have to generate all possible routes and takes the shortest. This becomes ractical as the number of towns, N, increases since the number of possible routes is (N-1)! The most effective exact algorithms are cutting –plane or facet –finding algorithms. These algorithms are quite complex, and are very demanding of computer power [3].

2.2.2. Approximation (or Heuristic) Algorithms

Heuristic algorithms generally suggest some approximations to the solution of optimization problems. These algorithms are able to produce an acceptable solution to a problem of the travelling salesman, but for which there is no formal proof of its correctness. Alternatively, it may be correct, but may not be proven to produce an optimal solution. Two fundamental goals in computer science are finding algorithms with provably good run times and with provably good or optimal solution quality [26].

A heuristic is an algorithm that abandons one or both of these goals; for example, it usually finds pretty good solutions, but there is no proof the solutions could not get arbitrarily bad; or it usually runs reasonably quickly, but there is no argument that this will always be the case. Every TSP heuristic solution can be evaluated in terms of two key parameters: its running time and the quality of the tours that it produces.These algorithms are much faster and they obtain good solutions, but they do not guarantee the optimal solution. Some of them give solutions that on average differ only by a few percent (2-3%) from the optimal solution. Therefore, if a small deviation from optimum can be accepted, it may be appropriate to use an approximation algorithm [3].

The heuristic algorithms can be categorized into the following three classes:

• Tour construction algorithms

• Tour improvement algorithms

• Composite algorithms

2.3 Special Method for TSP:

• Metric TSP

• Euclidean TSP

• Asymmetric TSP

2.3.1 Metric TSP

In the metric TSP, also known as delta-TSP or Δ-TSP, the intercity distances satisfy the triangle inequality. A very natural restriction of the TSP is to require that the distances between cities form a metric to satisfy the triangle inequality; that is the direct connection from *A* to *B* is never farther than the route via intermediate *C*:

The edge spans then build a metric on the set of vertices. When the cities are viewed as points in the plane, many natural distance functions are metrics, and so many natural instances of TSP satisfy this constraint.

The following are some examples of metric TSPs for various metrics.

• In the Euclidean TSP (see below) the distance between two cities is the Euclidean distance between the corresponding points

• In the rectilinear TSP the distance between two cities is the sum of the absolute values of the differences of their *x*- and *y*-coordinates. This metric is often called the Manhattan distance or city-block metric.

• Asymmetric TSP In the maximum metric, the distance between two points is the maximum of the absolute values of differences of their *x*- and *y*-coordinates.

The last two metrics appear for example in routing a machine that drills a given set of holes in a printed circuit board. The Manhattan metric corresponds to a machine that adjusts first one co-ordinate, and then the other, so the time to move to a new point is the sum of both movements. The maximum metric corresponds to a machine that adjusts both co-ordinates simultaneously, so the time to move to a new point is the slower of the two movements.

In its definition, the TSP does not allow cities to be visited twice, but many applications do not need this constraint. In such cases, a symmetric, non-metric instance can be reduced to a metric one. This replaces the original graph with a complete graph in which the inter-city distance is replaced by the shortest path between *A* and *B* in the original graph [3].

2.3.2 Euclidean TSP

When the input numbers can be arbitrary real numbers, Euclidean TSP is a particular case of metric TSP, since distances in a plane obey the triangle inequality. When the input numbers must be integers, comparing lengths of tours involves comparing sums of square-roots.

Like the general TSP, Euclidean TSP is NP-hard in either case. With rational coordinates and discretized metric (distances rounded up to an integer), the problem is NP-complete. With rational coordinates and the actual Euclidean metric, Euclidean TSP is known to be in the Counting Hierarchy, a subclass of PSPACE. With arbitrary real coordinates, Euclidean TSP cannot be in such classes, since there are uncountably many possible inputs.

However, Euclidean TSP is probably the easiest version for approximation. For example, the minimum spanning tree of the graph associated with an instance of the Euclidean TSP is a Euclidean minimum spanning tree, and so can be computed in expected O (n log n) time for n points (considerably less than the number of edges).

This enables the simple 2-approximation algorithm for TSP with triangle inequality above to operate more quickly.In general, for any $c > 0$, where d is the number of dimensions in the Euclidean space, there is a polynomial-time algorithm that finds a tour of length at most $(1 + 1/c)$ times the optimal for geometric instances of TSP in time; this is called a polynomial-time approximation scheme (PTAS) [3].

2.3.3 Asymmetric TSP

In most cases, the distance between two nodes in the TSP network is the same in both directions. The case where the distance from A to B is not equal to the distance from B to A is called asymmetric TSP. A practical application of an asymmetric TSP is route optimization using street-level routing (which is made asymmetric by one-way streets, slip-roads, motorways, etc.) [3].

Solving by conversion to symmetric TSP

Solving an asymmetric TSP graph can be somewhat complex. The following is a 3×3 matrix containing all possible path weights between the nodes A, B and C. One option is to turn an asymmetric matrix of size N into a symmetric matrix of size $2N$ *[3]*.

Asymmetric path weights

	A	***B***	***C***
A		1	2
B	6		3
C	5	4	

To double the size, each of the nodes in the graph is duplicated, creating a second ghost node, linked to the original node with a "ghost" edge of very low (possibly negative) weight, here denoted $-w$. (Alternatively, the ghost edges have weight 0, and weight w is added to all other edges.) The original 3×3 matrix shown above is visible in the bottom left and the transpose of the original in the top-right. Both copies of the matrix have had their diagonals replaced by the low-cost hop paths, represented by $-w$. In the new graph, no edge directly links original nodes and no edge directly links ghost nodes [2].

Symmetric path weights

	A	***B***	***C***	***A′***	***B′***	***C′***
A				$-w$	6	5
B				1	$-w$	4
C				2	3	$-w$
A′	$-w$	1	2			
B′	6	$-w$	3			
C′	5	4	$-w$			

[2]

2.4 Solving TSP using Nearest Neighbor Algorithm

The Nearest Neighbor Algorithm is an approximate algorithm for finding a sub-optimal solution to the TSP. In this algorithm, it starts with a city as a starting city and repeatedly visits all the cities until all the cities have been visited exactly once. It finds a shortest path, but solution is not the optimal one.

The algorithm of NN is:

Step 1: Start with any random vertex, call it current vertex.

Step 2: Find an edge which gives minimum distance between the current vertex and an unvisited vertex, call it V.

Step 3: Now set that current vertex to unvisited vertex V and mark that vertex V as visited.

Step 4: Terminate the condition, if all the vertices are visited at least once.

Step 5: Go to step 2 [20].

2.5 Knapsack Problem

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items [20].

The problem often arises in resource allocation where there are financial constraints and is studied in fields such as combinatiorics, computer science, complexity theory, cryptography, applied mathematics, and daily fantasy sports. The knapsack problem has been studied for more than a century, with early works dating as far back as 1897. The name "knapsack problem" dates back to the early works of mathematician Tobias Dantzig (1884–1956), and refers to the commonplace problem of packing the most valuable or useful items without overloading the luggage [20].

2.6 Assignment Problem

The assignment problem is one of the fundamental combinatorial optimization problems in the branch of optimization or operations research in mathematics. It consists of finding a maximum weight matching (or minimum weight prefect matching) in a weighted bipartite graph [17].

The problem instance has a number of agents and a number of tasks. Any agent can be assigned to perform any task, incurring some cost that may vary depending on the agent-task assignment. It is required to perform all tasks by assigning exactly one agent to each task and exactly one task to each agent in such a way that the total cost of the assignment is minimized [17].

If the numbers of agents and tasks are equal and the total cost of the assignment for all tasks is equal to the sum of the costs for each agent (or the sum of the costs for each task, which is the same thing in this case), then the problem is called the linear assignment problem. Commonly, when speaking of the assignment problem without any additional qualification, then the linear assignment problem is meant [17].

2.7 Nearest Neighbor Algorithm

The nearest neighbor algorithm is easy to implement and executes quickly, but it can sometimes miss shorter routes which are easily noticed with human insight, due to its "greedy" nature. As a general guide, if the last few stages of the tour are comparable in length to the first stages, then the tour is reasonable; if they are much greater, then it is likely that there are much better tours. Another check is to use an algorithm such as the lower bound algorithm to estimate if this tour is good enough [10].

In the worst case, the algorithm results in a tour that is much longer than the optimal tour. To be precise, for every constant r there is an instance of the traveling salesman problem such that the length of the tour computed by the nearest neighbor algorithm is greater than r times the length of the optimal tour [10].

Moreover, for each number of cities there is an assignment of distances between the cities for which the nearest neighbor heuristic produces the unique worst possible tour. (If the algorithm is applied on every vertex as the starting vertex, the best path found will be better than at least N/2-1 other tours, where N is the number of vertexes) [10].

2.8 Greedy Algorithm

A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage[1] with the intent of finding a global optimum. In many problems, a greedy strategy does not usually produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time [19].

For example, a greedy strategy for the travelling salesman problem (which is of a high computational complexity) is the heuristic: “At each step of the journey, visit the nearest unvisited city”. This heuristic does not intend to find a best solution, but it terminates in a reasonable number of steps; finding an optimal solution to such a complex problem typically requires unreasonably manu steps. In mathematical optimization, greedy algorithms optimally solve combinatorial problems having the properties of matroids, and give constant-factor approximations to optimization problems with submodular structure [19].

The Greedy heuristic can be implemented to run in time Θ(N2 log N) and is thus somewhat slower than Nearest Neighbor. On the other hand, its worst-case tour quality may be somewhat better. Base on this fact, the greedy algorithm could be very good candidate to be used for TSP calculations with big amount of cities which have to be computed in short time [19].

2.9 Genetic Algorithm

The terms like technology, development and the information revolution plays vital role in our world and led to increasing the speed of delivery, performance and excellence in work. It also helps to solve more problems and provide numerous methods to solve problems of decision making, classification problems and optimization problems [14].

The development and use of optimization model is pretty conventional. In some fields of economic analysis when the problem is huge, the use of optimization models has been restricted that also leads to large number of nonlinear influences. In order to find out the solution to the linear models approximation or simplification usage is necessary in most cases.

All these factors helped to create a science that facilitates the fastest and best solution among the set of solutions that may occur in the thinking of human which called the science of Genetic Algorithm (GA).Genetic Algorithm is a procedure based on search to compute exact or approximate solutions and search problems which is the alternative of traditional optimization technique. Genetic Algorithms are most appropriate for complex nonlinear models where locating the global and best possible solution is a difficult task. Genetic Algorithm is a useful approach potentially because of its techniques [14].

GA tag four main stages: evolution, selection, crossover and mutation. Based on the significant optimization or search criteria, the evolution procedure measures the fitness of each individual elucidation in the population and assigns it a qualified value. For development of the next generation, the course of action in selection procedure randomly selects individuals of the current population [14].

A range of substitute procedures have been projected but all pursue the idea that the fittest strategy has a greater chance of choice. The course of action in crossover takes two chosen individuals and combine them about a crossover point (one, two or uni form) thus creating two new individuals. The course of action in mutation method randomly transforms the genes of an individual focus to a small mutation factor, launching further arbitrariness in the population.

There are many problems solved in GA, such as Travelling Salesman, Bin Packing, 8 queens, IC board stung, Water Sprinkler, Gun firing, maze solving, cutting fabric and so on. The focus of this paper is Travelling Salesman Problem as one of the problem solved using GA [14].

Travelling Salesman Problem TSP is a given set of cities and distance between them which requires finding the shortest path of visiting each city only once. As mentioned earlier GA follows steps to solve any problem one of which is called crossover which has three types such as one point, two point and uniform. In TSP two points i.e. 1 order crossover is used because condition for the problem is satisfied and duplication is avoided in combination of possibilities [14].

A crossover point is preferred on the basis of the parents. A direct swap would introduce duplicates and remove necessary candidates from the list, given that the individual is an ordered list. So when the GA steps are followed for all generation in each experiment, the same genes of the parents is stabilized individually as named static crossover [14].

The Traveling Salesman Problem (TSP) is generally calculated in the field of Computer Science. The TSP was first defined around 150 years ago but emerged as a complex problem in the late 1940’s. Even after half a decade of research, the TSP has not been completely solved and appeared as the interests of computer scientists and mathematicians. The TSP can be applied to crack many handy problems considered day after day. Thus, a way out to the TSP would be extremely valuable. When an explanation to the TSP is conversed, it is about so many different and vibrant solutions one of which is discussed here with new approach potentially because of its techniques [14].

Typically Travelling Salesman problem is declared as a dilemma when a salesman need a trip of some given cities one time, initiating from any city and revisiting to the original place of departure. So the question arise that what itinerary should he opt in order to decrease the total distance traveled [14].

The difficulty was to get the shortest possible route once when the salesmen need to stopover their clients and end where they begin. This same problem now applies to a huge number of activities and forms the basis of solution to these modern day issues. For instance, distributing new stock to super marts, providing manufacturing lines, sky travel control, and even chromosome progression. The hypothetical impact on computer science and operational investigation also require TSP solution. Complex and sophisticated computer programmers using optimization where algorithms generate the best possible consequence from several alternatives.

The time required to find an best possible solution is essential for realistic application of the TSP. Matters like time required for lorry drivers to wait for their route to be finalized to deliver the salads as the salads will only be fresh for another 24 hours, Time required for air traffic to keep controlling an airliner flying in circles around Airport is studied in the theory of computational complication. For this area of study, the TSP has vital impact. It is of more interest and benefit to the scientific community even a small incremental pace in considering the nature of this problem [14].

Genetic Algorithms solve TSPs and provide quick solutions as they operate strings of information. Biological and Computer fields are used to describe GAs as they were stimulated by the conduct of natural systems. In past, using GAs, optimal solutions to TSPs were first formed by Goldberg using Partial Mapped crossovers and also using Greedy Crossover by Grefenstette. Using various crossovers operation, Davis, Smith, Suh and Van Gucht solved TSPs [14].

Genetic Algorithms imitate the technicalities of ordinary selection by a method of randomized data substitution. They are able to seek out in a randomized, directed way to allow them to imitate some of the novel potential of natural systems. The major concern for this paper is to introduce new technology for TSP that is based on some programming. The vision is to put up a program in shell program using two point crossover to solve TSP in new format. Experiments are performed in which static and dynamic crossover both are used in order to find optimal solution. Execution time is computed and result is shown in the last section of this paper to prove that the time for dynamic crossover is less than that of static so noticeably dynamic crossover verifies better result than the other [14].

This paper proposes a solution to TSP using GA, dynamic crossover that uses the two point (1 order) crossover that changes the stabilized genes in each generation for all experiments. Later compute execution time for static and dynamic crossover in order to compare the improvements in finding the best solution. Previously, many researches had proposed and many algorithms and enhancement approaches for solving the TSP. the crossover technique might be different form problem to another, depended on the problem itself.

To describe different types of crossover operators, let’s start with the sequence described by Davis and Goldberg for the Order Crossover operator. The offspring become heir to the elements linking the two crossover points that selected from parents in the same order and position.

In the order in which the left over elements emerged are inherited from the alternate parent. The crossover points are determined then swapping the other elements in the parent. Delete the cities that are already exit because there is no need of redundancy cities. The new string called the offspring that used in next population. Also this technique is developed by Sysweda [14].

Also there is Partially Mapped Crossover (PMX) that is described by Goldbreg and Lingle. A parent and two crossover positions are selected randomly, cut two substrings of equal size on each parent at the same position and exchange between them to produce part of offspring, and then determine the mapping points based on selected substrings. Finally check if there redundancy cities substitute from other parent. Note that when substitution occurs, it results in a mutation where adjacency, position, or relative order is preserved by substitution. Also note that PMX is dependent on crossover position [14].

Finally, the last type called Cycle Crossover (CX) that proposed by Oliver, it is built offspring from one parent that is distinguished from other types. An element point is randomly selected and started as a cycle from a selected parent subset. The element in which the same position in other parent cannot be swapped in this position because it is produced the same subset. Follow this technique until the cycle is completed and if there is any element not presented in the offspring, selected it from unselected parent. So the offspring is always permutation [14].

The various forms of crossover and mutation can be combined to provide an assortment of genetic algorithms that can be used to crack the travelling salesman problem. Greedy algorithms are one of the methods for finding practical solution to the travelling salesman problem. The algorithm generates a list of all edges in the graph and then arrange them from smallest cost to largest cost after which it chooses the edges with smallest cost first. The greedy algorithm gives feasible services [14].Many methods based on crossover have been implemented for TSP. The

Travelling Salesman Problem is a problem in combinatorial optimization solution in short amount of time is required. With the particular set of cities and their pair-wise distances, scheme is to find a undeviating tour that each city is visited just one time with the length of the tour to be minimized [14].

Much less time is used to find a solution by means of a genetic algorithm. It can find a near perfect solution for a city tour in less than specific time even though it might not find the best solution. There are a few basic steps to solve the travelling salesman problem using.

Step 1: Form a crowd of many random tours that is known as population. Here use a greedy initial population that gives preference to connecting cities that are close to each other.

Step 2: Pick two better or shorter tours parent in the population and combine them to make two new child tours. Hopefully, these children tour will be better than any of the parent [14].

2.10 Simulated annealing

Simulated annealing has been invented in 1983 and as described in, briefly written it uses an approach similar to Nearest Neighbor/Hill-climbing, but occasionally accepts solutions that are worse than the current. The probability of such acceptance is decreasing with time [9].

The reason for this behavior can be clear from the original invention – the name "simulated annealing" derives from the intent to pattern the approach after the physical process of annealing, which is a means for reducing the temperature of a material to its low energy or ground state. Such a state may be viewed as analogous to an optimum, where the energy level may accordingly be viewed as the value of an objective function to be minimized [9].

The annealing process begins with a material in a melted state and then gradually lowers its temperature, analogous to decreasing an objective function value by a series of improving moves. However, in the physical setting the temperature must not be lowered too rapidly, particularly in its early stages. Otherwise certain locally suboptimal configurations can be frozen into the material and the ideal/optimal low energy state will not be reached. To allow a temperature to move slowly through a particular region corresponds to permitting non-improving moves to be selected with a certain probability – a probability which diminishes as the energy level (objective function value) of the system diminishes.

Thus, in the analogy to combinatorial problem solving, it is postulated that the path to an optimal state likewise begins from one of diffuse randomization, somewhat removed from optimality, where non-improving moves are initially accepted with a relatively high probability which is gradually decreased over time [9].

2.11 Tabu search

Tabu search extends the idea of simulated annealing to avoid local optima by using memory structures. The problem of simulated annealing is that after "jump" the algorithm can simply repeat its own track. Tabu search prohibits the repetition of moves that have been made recently [5].

Tabu search belongs to the most effective solution techniques for solving hard combinatorial problems. Originally proposed by Glover, it has been successfully applied to a wide variety of application contexts, such as vehicle routing/TSP, machine scheduling, maximum clique problem, quadratic assignment problem or the nurse scheduling problem [5].

Generally speaking, tabu search is a local search technique, i.e., an iterative search procedure that, starting from an initial feasible solution, progressively improves it by applying a series of local modifications. The key ingredient of any local search technique is the set of modifications (or moves) that it considers: the richer this set, the better the solutions that one can expect to obtain, but also the slower the method [5].

While classical local search methods stop when they encounter a local optimum with regard to the modifications they allow, tabu search continues moving to the best non-improving solution it can find. Cycling is prevented through the use of short-term memory structures called tabu lists [5].

2.12 Dynamic Programming

Dynamic programming is both a mathematical optimization method and a computer programming method. The method was developed by Richard Bellman in the 1950s and has found applications in numerous fields, from aerospace engineering to economics. In both contexts it refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner. While some decision problems cannot be taken apart this way, decisions that span several points in time do often break apart recursively [20].

Likewise, in computer science, if a problem can be solved optimally by breaking it into sub-problems and then recursively finding the optimal solutions to the sub-problems, then it is said to have optimal substructure.If sub-problems can be nested recursively inside larger problems, so that dynamic programming methods are applicable, then there is a relation between the value of the larger problem and the values of the sub-problems. In the optimization literature this relationship is called the Bellman equation.

Dynamic programming (usually referred to as DP. Also, the optimal solutions to the subproblems contribute to the optimal solution of the given problem. DP) is a very powerful technique to solve a particular class of problems. It demands very elegant formulation of the approach and simple thinking and the coding part is very easy [20].

If solving a problem with the given input, then save the result for future reference, so as to avoid solving the same problem again. If the given problem can be broken up in to smaller sub-problems and these smaller subproblems are in turn divided in to still-smaller ones, and in this process, if you observe some over-lappping subproblems, then it is a big hint.

There are two ways of doing this.

1. Top-Down: Start solving the given problem by breaking it down. If you see that the problem has been solved already, then just return the saved answer. If it has not been solved, solve it and save the answer. This is usually easy to think of and very intuitive. This is referred to as Memoization.

2. Bottom-Up: Analyze the problem and see the order in which the sub-problems are solved and start solving from the trivial subproblem, up towards the given problem. In this process, it is guaranteed that the subproblems are solved before solving the problem. This is referred to as dynamic programming [20].

2.13 Branch and bound

Branch and bound is by far the most widely used tool for solving large scale NP Hard combinatorial problems. It is, however, an algorithm paradigm, which has to be filled out for each specific problem type, and numerous choices for each of the components exist [6].

Branch and bound technique is a critical enumeration of the search space. It enumerates, but constantly tries to rule out parts of the search space that cannot contain the best solution, by using upper and lower estimated bounds of the quantity being optimized [6].

A Branch and bound procedure requires two tools. The first one is a splitting procedure that, given a set *S* of candidates, returns two or more smaller sets S1, S2,.. whose union covers *S*. Note that the minimum of f(x) over *S* is min{*v1*,*v2*,...}, where each *vi* is the minimum of f(x) within Si.

This step is called branching, since its recursive application defines a search tree structure whose nodes are the subsets of *S*. Another tool is a procedure that computes upper and lower bounds for the minimum value of f(x) within a given subset *S*. This step is called bounding [6].

The key idea of the Branch and bound algorithm is: if the lower bound for some tree node (set of candidates) *A* is greater than the upper bound for some other node *B*, then *A* may be safely discarded from the search. This step is called *pruning*, and is usually implemented by maintaining a global variable *m* (shared among all nodes of the tree) that records the minimum upper bound seen among all subregions examined so far.

Any node whose lower bound is greater than *m* can be discarded. By implementing this tool a lot of possible routes is not taken into account and it's obvious that the requirements on the time for the computation are boosted by this tool and so that it speeds up the optimal route generation significantly compared to brute-force.

The recursion stops when the current candidate set *S* is reduced to a single element; or also when the upper bound for set *S* matches the lower bound. Either way, any element of *S* will be a minimum of the function within *S*. Despite the time complexity of branch and bound is exponential, it is typically much faster, sometimes thousands of times faster [6].

Thus many problems that cannot be done by the naive brute-force algorithm are easily done by branch and bound. For this reason, it is used to solve the optimization problems like satisfiability, TSP, integer programming and non-linear programming. These facts promote this exact algorithm to be very potential candidate to be used for the TSP computation in this work [6].

2.14 Branch and Cut

Branch and Cut algorithm is a combination of Cutting planes and Branch and Bound 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 further 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 linear program, such that resolving it will yield a different solution which is hopefully "less fractional" [18].

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. At this point, the Branch and Bound part of the algorithm is started. The problem is split into two versions, one with the additional constraint that the variable is greater than or equal to the next integer greater than the intermediate result, and one where this variable is less than or equal to the next lesser integer. In this way new variables are introduced in the basis according to the number of basic variables that are non-integers in the intermediate solution but which are integers according to the original constraints [18].

The new linear programs are then solved using the simplex method and the process repeats until a solution satisfying all the integer constraints is found. During the branch and bound process, further cutting planes can be separated, which may be either global cuts, i.e., valid for all feasible integer solutions, or local cuts, meaning that they are satisfied by all solutions fulfilling the 15 side constraints from the currently considered branch and bound sub-tree [18].

2.15 Distance Measure

Distance is a numerical measurement of how far apart objects are. In physics or everyday usage, distance may refer to a physical length or an estimation based on other criteria [12].

2.15.1 Similarity Distance Measure

The similarity measure is the measure of how much alike two data objects are. Similarity measure in a data mining context is a distance with dimensions representing features of the objects. If this distance is small, it will be the high degree of similarity where large distance will be the low degree of similarity.

The similarity is subjective and is highly dependent on the domain and application. For example, two fruits are similar because of color or size or taste. Care should be taken when calculating distance across dimensions/features that are unrelated. The relative values of each element must be normalized, or one feature could end up dominating the distance calculation. Similarity are measured in the range 0 to 1 [0,1].

Two main consideration about similarity:

- Similarity = 1 if X = Y (Where X, Y are two objects)
- Similarity = 0 if $X \neq Y$ [12].

2.15.2 Euclidean Distance

Euclidean distance is the most common use of distance. In most cases when people said about distance, they will refer to Euclidean distance. Euclidean distance is also known as simply distance. When data is dense or continuous, this is the best proximity measure.

The Euclidean distance between two points is the length of the path connecting them. The Pythagorean theorem gives this distance between two points [12].

2.15.3 Manhattan Distance

Manhattan distance is a metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates. In a simple way of saying, it is the total sum of the difference between the x-coordinates and y-coordinates [21].

Suppose we have two points A and B if we want to find the Manhattan distance between them, just we have, to sum up, the absolute x-axis and y – axis variation means we have to find how these two points A and B are varying in X-axis and Y- axis. In a more mathematical way of saying Manhattan distance between two points measured along axes at right angles.

In a plane with p1 at (x1, y1) and p2 at (x2, y2).

Manhattan distance = $|x1 - x2| + |y1 - y2|$

This Manhattan distance metric is also known as Manhattan length, rectilinear distance, L1 distance or L1 norm, city block distance, Minkowski's L1 distance, taxi-cab metric, or city block distance [21].

2.15.4 Cosine similarity

Cosine similarity metric finds the normalized dot product of the two attributes. By determining the cosine similarity, we would effectively try to find the cosine of the angle between the two objects. The cosine of 0° is 1, and it is less than 1 for any other angle.

It is thus a judgement of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude.

Cosine similarity is particularly used in positive space, where the outcome is neatly bounded in [0,1]. One of the reasons for the popularity of cosine similarity is that it is very efficient to evaluate, especially for sparse vectors [21].

2.15.5 Haversine Formula

The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the law of haversines, that relates the sides and angles of spherical triangles [21].

These names follow from the fact that they are customarily written in terms of the haversine function, given by haversin(θ) = $\sin^2(\theta/\boxed{2})$. The formulas could equally be written in terms of any multiple of the haversine, such as the older versine function (twice the haversine). Prior to the advent of computers, the elimination of division and multiplication by factors of two proved convenient enough that tables of haversine values and logarithms were included in 19th and early 20th century navigation and trigonometric texts. These days, the haversine form is also convenient in that it has no coefficient in front of the $\sin^2$ function [21].

Calculate geographic distance on earth. If you have two different latitude – longitude values of two different point on earth, then with the help of Haversine

Formula, you can easily compute the great-circle distance (The shortest distance between two points on the surface of a Sphere).

Haversine is very popular and frequently used formula when developing a GIS (Geographic Information System) application or analyzing path and fields.

To derive law of Haversine one needs to start the calculation with spherical law of cosine i.e $\cos a = \cos b * \cos c + \sin b * \sin c * \cos A$ .One can derive Haversine formula to calculate distance between two as:

$a=\sin^2(\Delta latDifference/2)+\cos(lat1).\cos(lt2).\sin^2(\Delta lonDifference/2)$

$c=2.atan2(\sqrt{a},\sqrt{(1-a)})$

$d = R.c$

where,

$\Delta latDifference = lat1 – lat2$ (difference of latitude)

$\Delta lonDifference = lon1 – lon2$ (difference of longitude)

R is radius of earth i.e 6371 KM or 3961 miles [21].