Solving the TSP using Traditional Computing Approach

Article *in* International Journal of Computer Applications · October 2016

DOI: 10.5120/ijca2016911906

2 authors:

Evans Baidoo
Hohai University
22 PUBLICATIONS **101** CITATIONS
SEE PROFILE

Stephen Opoku Oppong
University of Education, Winneba
29 PUBLICATIONS **169** CITATIONS
SEE PROFILE

Solving the TSP using Traditional Computing Approach

Evans Baidoo
Kwame Nkrumah University of Science and Technology
Department of Computer Science

Stephen O. Oppong
Kwame Nkrumah University of Science and Technology
Department of Computer Science

ABSTRACT
From the last decade, even though there have been sudden advances in present technology in all areas, there exist some real-world NP composite problems that still escape scientists. The Travel salesman Problem is no exception. As it is an NP-Hard problem, lots of divergent solutions have been created to determine in shortest possible time, the optimal solution. Traditional algorithms are one of the oldest suggested solutions which present successful solutions that are to a larger extent optimal except in few occasions which may be close to the optimal. In this paper, a variant of the classical TSP, Random TSP (RTSP) is computed using various traditional algorithms. Their performances are evaluated with emphasis on length of tour and the algorithm effectiveness. Also, this paper presents the comparison among the algorithms based on a variety of parameters that facilitated to decide the superior algorithm with regards to their needs.

General Terms
Travelling Salesman Problem, Dynamic Programming, Branch and Bound and Nearest Neighbor algorithm

Keywords
Traditional Algorithms, Travelling Salesman Problem, Optimization Problem

1. INTRODUCTION
Travelling Salesman Problem (TSP) is classical and most widely studied problem in Combinatorial Optimization [1]. It has been studied intensively in both Operations Research and Computer Science since 1950s as a result of which a very large amount of methods were studied to solve this problem. The study of TSP presents a perfect platform for study of general methods that can be applicable to a broad range of Discrete Optimization Problems so the need to study but not really motivated by direct applications . Certainly, several direct applications of TSP breathe life to research area and help out to direct future work. The idea of the problem is to find shortest route of salesman starting from a given city, visiting n cities only once and finally arriving at origin city. The problem can be sketched on graph with each city becoming a node. Assuming a complete weighted graph, edge lengths correspond to the distance between the attached cities.

The TSP occurs in countless forms with some applications of engineering that include Vehicle routing [2] scheduling problems [3], integrated circuit designs [4], physical mapping problems [5], and constructing phylo-genetic trees [6].

There are a lot more algorithms known to have been developed to solve TSPs where many algorithms were applied with more or less success. On the order of merits there are diverse ways to classify algorithms. The implementation principle is one of the ways to categorize algorithms and is by [7].

Explicit enumeration: It leads to investigating all feasible solutions of problems, for that reason is appropriate only for small problem size (Brute Force, Greedy approach, Divide and conquer etc). It is frequently used in situations where no analytical solution algorithms exist and the solution space is finite.

Deterministic methods: These algorithms base only on exact methods of "classical" mathematics. It is simply an algorithm that has a predefined output. With a given specific input, it will at all times return the same output. Added information, such as gradient, convexity etc. is as a rule needed. Examples may include Branch and Bound Algorithm, Cutting Plane Method, Dynamic Programming etc. Deterministic algorithms are by far the most studied and familiar kind of algorithm and as well the most practical, due to their efficient running on real machines.

Stochastic methods: Stochastic approach is one in which values are attained from a matching chain of jointly scattered random variables. These algorithms work on probabilistic methods to answer problems. The algorithm work gradually and are generally applicable only for guessing (Evolutionary Computation, Random search Walk, Monte Carlo etc.).

Combined methods: These methods generally consist of stochastic and deterministic composition. Several meta-heuristics algorithm has been formulated (Ant Colony Optimization, Memetic Algorithms, Genetic algorithms, Tabu-search, Simulated Annealing, Firefly Algorithm etc.). Meta-heuristics is made up of general search processes whose principles permit them to escape local optimality by means of heuristics design.

In this research paper, among other criteria the authors classify the explicit enumeration and deterministic methods as traditional methods due to their year of discovery and implementation and a less emphasized fact that both methods have the ability to require any auxiliary information and reject probabilities in their operators. Additionally they work with single points as opposed to working with population of points and on variables not string coding of variables.

This paper take a look at two explicit enumeration algorithms and three deterministic algorithms based on their qualities of their solutions and mechanisms by which edges that emerge in a known optimal tour are conserved and summed to produce the minimized and optimized tour length. The rest of the paper is structured as follows: Section 2 details explanation of the Travelling Salesman Problem; Section 3 in brief examines all five algorithms used in this work. Experimental results are revealed in Section 4 and finally in Section 5 the conclusion of the work.

2. TRAVELLING SALESMAN PROBLEM
The origin of the Travelling Salesman Problem and its name is to some extent difficult to understand. .The travelling

salesman problem, also recognized as the TSP, is one of the problems in computer science which is largely prominent. Although it is simple to explain, yet it is very complicated to compute. It emerges to have been talked about informally among mathematicians for quite a lot of years. The TSP is a known combinatorial optimization problem. Even though many argue to solve this problem to the optimal within efficient time, there is no known algorithm which can solve it completely in a reasonable amount of time. Although many argue it is difficult to solve, we settle on using traditional methods to solve this problem and to conduct study into finding among the traditional approaches the most efficient offering optimal results.

TSP first appearance in mathematical theory is the paper by Leonard Euler in 1757. Euler's paper concerns a resolution of the knight's tour problem in chess, that is, the problem of locating a series of knight's movement that will take the piece from a starting square on a chessboard, through every other square precisely once and returning to the start. This is classified as the Classical Travel Salesman Problem (CTSP). With the CTSP, as the quantity of cities raises, the complexity of the problem also raises exponentially because the number of promising solutions increase very much.

The test of the CTSP is that the travelling salesman wants to reduce the total length of the trip.

The travelling salesman problem can be illustrated [16] as follows:

TSP = {(G, f, t): G = (V, E) a complete graph,

f is a function $V \times V \rightarrow Z$,

$t \in Z$,

G is a graph that contains a travelling salesman tour with cost that does not exceed t}.

A look at the subsequent set of cities in fig 1.1:

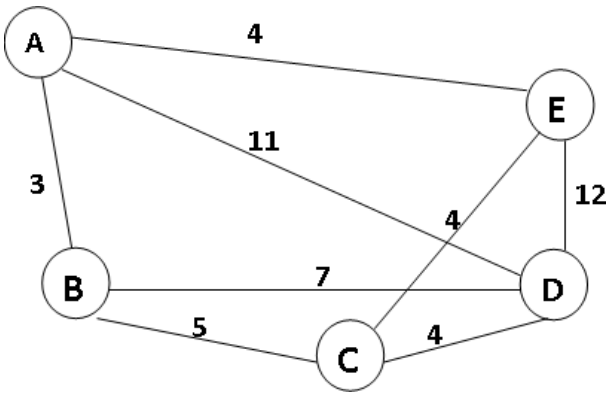

Figure 1.1: A graph with weights on its edges.

The goal of the problem lies in identifying a minimum length passing from all n nodes of G vertices exactly once. For instance the path Path1 {A, B, C, D, E, A} and the path Path2 {A, B, C, E, D, A} go by all the vertices however Path1 has a sum length of 28 and Path2 has a sum length of 35. P = {A, B, C, D, E} forms a Hamiltonian cycle in view of the fact that each vertices is traversed once. As a result, an optimal solution to TSP is permutation π with node indices {1,.....,n} such that length f(π) is minimal, where f(π)is given by [8],

$$F(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)}$$

This paper offers solutions to a variation of TSP referred to as Random Travelling Salesman Problem (RTSP) using classical explicit enumeration and deterministic algorithms. Algorithm performances are evaluated on the basis of length of tour and the effectiveness of the algorithm. Four datasets are arbitrarily created having the number of cities and coordinates as the distances for cities and represented in the form of adjacency matrix. The generated coordinates have a limited range from 0 to 100. Mostly, a starting node continues with it if is specified, if not starting node is selected arbitrarily.

3. TRADITIONAL ALGORITHMS

Two classical explicit enumeration (Brute force, and Greedy method), and three deterministic methods (Dynamic Programming, Branch and Bound and Nearest Neighbor algorithm) have been used for solving the TSP alongside their implementation comparison in this work. These are in brief illustrated along with their algorithms in the succeeding section.

3.1 Brute Force

Brute force is an exhaustive search of all the probable results for a problem. It is a programming approach that does not embrace any shortcuts to advance performance, but instead relies on complete computing power to attempt all possibilities until a reliable solution to a problem is established of which a classic example is the traveling salesman problem (TSP). It is a lot easy to execute and will almost definitely find a solution (If there is one). In terms of Algorithmic complexity, it is time consuming. Even though brute force algorithm is not for the most part elegant, it does have a justifiable place in software engineering. Since brute force methods at all times return the correct result -- although slowly -- they are practical for testing the correctness of faster algorithms. Additionally, occasionally a particular problem can be solved so quickly with this algorithm that it will be out of place to waste time devising a more promising solution.

Algorithm according to [9]

1. *Create a record of all promising Hamilton circuits*
2. *Compute the weight or distance of every Hamilton circuit by totalling the weights or distance of its edges.*
3. *Choose the Hamilton circuit with the smallest total weight.*

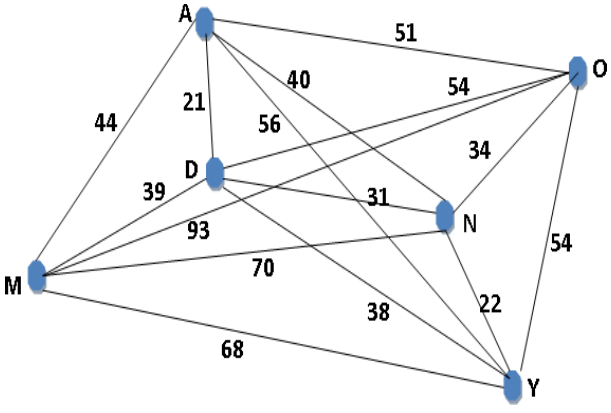

Figure 1.2: A TSP instance

From fig 1.2, which sequence should be travelled to ensure that minimum distance is covered?

Table 1: Distance between each pair of cities (Km)

	DISTANCE COVERAGE					
CITIES	M	O	N	D	A	Y
M	0	93	70	39	44	68
O	93	0	34	54	51	54
N	70	34	0	31	40	22
D	39	54	31	0	21	38
A	44	51	40	21	0	56
Y	68	54	22	38	56	0

Solving some possible outcome of distances to be covered between cities

Distance (MDNYOAM) = 39 + 31 + 22 + 54 + 51 + 44 = 241

Distance (MNAYDOM) = 70 + 40 + 56 + 38 + 54 + 93 = 351

Distance (MONDAYM) = 93 + 34 + 31 + 21 + 56 + 68 = 303

From the example given the number of Vertices (Cities) is N = 6. It pans out that there are precisely 5! = 5 * 4 * 3 * 2 * 1 = 120 dissimilar permutations of the numbers to n-1from 0.As the attention relies on permutations that begin with 0, to compute an n-city TSP instance with brute force necessitates that we resolve at exactly (n – 1)! = 1. 2. 3….. (n – 2), (n – 1) different permutations. The brute force algorithm can be used on small city problems as it is O(n!).

3.2 Greedy Method

Local optimization happens to be the key motive behind a greedy algorithm. This implies that the algorithm selects at one time what appears to be the important thing to do, instead of considering the global situation. In other situations, whilst the optimal solution is too costly, a greedy algorithm could be able to come up with an OK solution. Greedy algorithm basically attempts to do what appears like the short-term best thing, and hopes that this pans out in the long run. For the most times, greedy algorithms are easy to invent, simple, easy to implement and straightforward approach. It more often than not does not operate comprehensively on all the data but may offer locally optimal solutions that estimate a global optimal solution in a logical time.

Finding an optimal way to a solution, the greedy method constructs two set where one set keeps accepted items and the other containing rejected items. The greedy algorithm comes up with some five (5) components [10].

1. A function that verifies that a selected set of items present a solution.
2. A function that looks out for the feasibility of a set.
3. A selection function that identifies the most promising amongst candidates
4. An objective function, which does not emerge explicitly, provides the value of a solution.
5. A solution function, which will indicate when a complete solution is discovered

Algorithm as stated by [11]

- Before stating the lemma, we need some notation and preliminary concepts. *Let*

  V(G) and E(G) be the vertex and edge sets of G,

  Assume G = (V, E) is the graph given, with | V| = n

  {

  Begin graph with T = (V,∅) making up of vertices of G only and no edges;

  Set up E in increasing order of costs;

  ***for** (i = 1, i ≤ n - 1, i + +)*

  {

  Pick the next least cost edge;

  ***if** (the edge joins two dissimilar connected components)*

  add the edge to T;

  }

  Return T

  }

  From the example 1.0

- *Find the shortest path among the cities (n)*

- *C ← {} // set that will keeps the cities*
- *Sol ← {}; // set that will hold the solution set.*
- *Sum ← 0 addition of item in solution set*
- *Sort the edges in the increasing order of weights.*
- *Starting with the least cost edge*
- *WHILE sum != n*
- *Scale through the edges one by one*
- *Select an edge only // Check for the constraint*
  - *IF the edge, together with already picked edges, does not make a vertex to possess degree three or more AND*
  - *does not form a cycle, unless the number of chosen edges is the same the number of vertices in the graph.*
- *ELSE RETURN "No Solution"*
- *x = shortest distance in set C such that sum + x ≤ n*
- *IF item does not exist THEN*

- *RETURN "No Solution"*
- *Sol ← S {value of x}*
- *sum ← sum + x*
- *RETURN Sol*
- *END*

From Fig 1.2, without loss of generality, the greedy method will return the results as follows:

Distance (MDAONYM) = 39+21+51+34+22+68 = 235

Distance (MDNAOYM) = 39+31+40+51+54+68 = 283

Distance (MDANOYM) = 39+21+40+34+54+68 = 256

The results form a Hamilton circuit. Therefore the greedy method will pick from among these possible candidates the optimal/ most promising solution set. i.e. the set that returns the least cost.

3.3 Branch and Bound

There were at least three groups that independently discovered the Branch and bound technique. It was first applied by [12] to solve asymmetric TSP. This exceptionally noteworthy paper also introduced quite a lot of other innovations. Land and Doig [13] provided a self explanatory general description to solving integer programming problems by linear programming. Little [14] finally described and named this approach as Branch and Bound in an application to the TSP. Branch and bound is a method that uses a state space search where expansion of any of the children may take place when all the sub-problems of a node are created. Although similar to backtracking method it employs a breadth first search algorithm-like search.

The technique adopted by this method is to divide a problem into a number solvable sub-problem. It solves a series of sub-problems of which each may have numerous possible solutions and where the chosen sub-problem for one solution may influence the possible solutions of later sub-problems.

A branch-and-bound algorithm comprises of a complete computation of all node solutions, where large subsets of ineffective nodes are discarded, by means of upper and lower approximated bounds of the optimize quantity. To keep away from the complete computation of all partial graphs, a practical solution is first found and its value noted as an upper bound for the optimum. Computations are done as the distance exceeds the upper bound. The value of a new found cheaper solution is used as the new upper bound when found.

Principle

Assuming an objective function is required to be minimized and assuming that there is a method for obtaining a lower bound on the cost of whichever solution amongst those in the set of solutions corresponding to some subset. If the subset with the best solution found so far costs less than the lower bound, then exploring that subset further is aborted.

Let S represent some subset of solutions.

L(S) = a lower bound on the cost of whichever solution belonging to S

Let C = best solution cost found so far

If $C \leq L(S)$, abort exploring S since it does not have any better solution.

If $C > L(S)$, explore S further since it may have a better solution

Algorithm

```
/* survive_node_set: set to keep the surviving nodes at all
time */
/* low-cost: variable to keep the minimum cost of the cost at
any set node */
Start
        Low-cost = ∞;
        While survive_node_set ≠∞ do
                - select a branching node, q, in that
                q ∈ survive_node_set; /* q is a E-node */
                - survive_node_set= survive_node_set -
        {q};
                - produce the offspring of node q and the
                equivalent lower bounds;
                S_q = {(j,Z_j): j is product of q and Z_j
                - For each element (j,Z_j) in S_q do
                        - If Z_j > T
                        - then
                                - kill product j; /* j is
                        a product node */
                        - Else
                                If product j is a
                        solution
                                Then
                                T =Z_j ; current best =
                        product j;
                                Else
                                Add product j to
                        survive_node_set;
                                Endif;
                        Endif;
                - Endfor;
        Endwhile;
```

3.4 Dynamic Programming

Dynamic programming (DP) is a very dominant technique that finds solution to a specific group of problems. It is an optimization technique that alters a series of simpler problems from a complex problem. It is a method for efficiently calculating recurrences by holding partial results and re-process them when the need be. The multistage character of the optimization method is its important characteristic. More so than the optimization approach as illustrated earlier, dynamic programming presents a broad framework for analyzing countless problem types. Within this framework an array of optimization procedures can be utilized to solve particular characteristics of a more universal formulation. It is a known fact that dynamic-programming cycle can be stated as shortest-path problems in a more layered networks whose nodes link up to the states of the dynamic program. More often than not creativity is essential before we can make out that a specific problem can be cast successfully as a dynamic program; and frequently subtle insights are needed to reorganize the formulation so as to solve it effectively. It requires very smart formulation of the problem and simple thinking.

There are two ways of arriving at an optimal solution of a given problem. This may be by Top down or Bottom up approach. The top down approach begin solving the given problem by breaking it down. If it identifies that the problem has been solved already, it just return the saved answer else it solve it and save the answer. This property is known as Memoization[15]. The Bottom up approach examines the

problem and notes the order the sub-problems are solved and begin solving from the smallest sub-problem, up to the given problem. In this procedure, it is ensured that the sub-problems are solved prior to solving the problem

Algorithm of the Dynamic programming as stated by [16]

//S=set of all cities, ℵ=number of cities, C = city/node
1. Select a random vertices/ node (city) as a first initial node Q
2. P=Power set of all node excluding Q or 2^{S-Q}
3. ***for*** *k=2 to* ℵ ***do*** *//create all permutation of node/cities*

g(k,∅)=Ck1 //initialization
4. ***for*** *all i* ϵ *S - {1}* ***do***
for *all element E in* P ***do***
if *i not in E* ***then***
g(i, E)= $min_{j \in E}(C_{ij} + g(j_1, E\text{-}\{j\}))$
//add to g shortest distance
5. $g(1,S\text{-}\{1\})=min_{j \in S-\{1\}}(C_{ij} + g(j,S\text{-}\{1\}\text{-}j))$

//shortest distance calculated

3.5 Nearest Neighbor Algorithm

The nearest neighbor algorithm happens to be among the earliest known algorithms used to settle on a solution to the travelling salesman problem.

J.G. Skellam pioneered the algorithm which was then continued by F.C. Evans and P.J Clark. The algorithm is such that, it arbitrarily selects a city as the starting city and then navigates to all neighboring cities closest to the staring city that does not form any cycle. This process is maintained until all cities are reached once.

In a related work, Taiwo et al [17] offered a reliable implementation approach, Nearest Insertion and Nearest Neighbor to finding solution to the TSP. In their work, a comparison was made to ascertain the algorithm that gives the superior result and to find out the flaws in the other algorithm with which it fails to produce the required result. Additionally, they compared the execution time and concluded that the nearest insertion algorithm has somewhat less time than that of nearest neighbor algorithm. From their observation the solution can be established in very limited computational time. This assertion to a larger extent facilitated to arrive at the conclusion that to obtain satisfactory results which may not be optimal but are close to the optimal result these approaches can be adopted.

Algorithm

1. *Pick any node to begin with*
2. *identify neighbouring nodes of the starting node which is not yet visited with the shortest distance.(if a tie is found, randomly break it)*
3. *At each stage, repeat this procedure to visit exactly once all nodes in the tour*
4. *Return to the starting nodes if all nodes are visited else continue with step 2*
5. *Compute the total minimum distance of the tour.*

This algorithm presents a chain of all the visited vertices but often time misses out some of the shorter routes.

4. EXPERIMENTAL RESULTS

All five labeled traditional algorithms are implemented with some adjustment in a small number of parameters in order to adjust it to work out the travelling Salesman problem. The experimental settings is executed in Java program and carried out on a HP ProBook 4540s Computer with the processor of Intel(R) Core(TM) i3-3110M CPU at 2.40 GHz and 4096 GB memory. Randomly generated TSP dataset of four instances are used to solve the travel salesman problem and to compare the effectiveness and performance of all five algorithms. Table 2 illustrates the performance comparison of all five traditional algorithms for the Random TSP whiles Table 3 compares their effectiveness.

Table 2: Performance Comparison

Algorithm	No of Cities	Best Distance results	Execution time
Brute Force	4	120	0.1245
	6	228	0.4564
	10	1635	3.6546
	12	2022	12.5648
Greedy Algorithm	4	120	0.1123
	6	254	0.2454
	10	1639	0.4745
	12	2045	0.6473
Branch and Bound	4	120	0.1143
	6	228	0.3471
	10	1635	0.6457
	12	2027	0.9874
Dynamic Algorithm	4	120	0.1134
	6	228	0.3418
	10	1635	0.6300
	12	2025	0.9684
Nearest Neighbour Algorithm	4	120	0.1120
	6	269	0.2281
	10	1639	0.4671
	12	2047	0.5754

Table 3: Effectiveness comparison

Algorithm	Feasible solution	Optimal result	Ease of implementation	Simplicity
Brute Force		√	√	√
Greedy Algorithm	√		√	√
Branch and Bound	√		√	
Dynamic Algorithm		√	√	√
Nearest Neighbour	√		√	√

From table 2, the Brute-Force Algorithm returns the best solution, but takes an unreasonably long time to compute than the other algorithms. The Greedy and Nearest Neighbor algorithms offer close to optimal results in reasonable limited time. The Branch and Bound and Dynamic programming produce very good results but may not always provide the best results. The Nearest Neighbor algorithm returns the best execution time.

From Table 3, Brute force and Dynamic programming algorithm indicate that it is optimal but inefficient. The Greedy algorithm, Branch and Bound and Nearest Neighbor indicated their efficiency or offering feasible solution but may fail to provide optimal solution.

On the whole from the tables 2 & 3, the dynamic programming algorithm is the best choice of solution to the travel salesman problem with the given set of conditions

5. CONCLUSION

Finding a sub-optimal solution to the TSP can be obtained using any of the five traditional algorithms; Brute force, Greedy algorithm, Branch and Bound, Dynamic programming and Nearest Neighbor. Making reference to the Greedy, Brute Force and Nearest neighbor, in the algorithm, each node match to their initial node taking into consideration the next closet node. This goes to emphasize the fact that nodes are not free of one another.

The paper offers a comparison among the traditional algorithm to solving the travelling salesman problem. The comparison criteria are rooted on the distance travelled by the algorithms, their execution time and their effectiveness.

In all four instances, the Brute force approach returned the best results but possess unreasonably high execution time whiles the execution time of Nearest Neighbor algorithm possess the least but returned un-optimal solution in all the four instances. The Dynamic programming algorithm on the other hand, returned optimal solution within a reasonable execution time, easy to implement and very simple. The major object of a TSP is to identify the lowest total distance travelled, so in view of the objective of TSP, it is concluded that the Dynamic programming algorithm is considered the best algorithm with regards to the criteria under consideration.

In future, further studies can be conducted on comparing the performance and cost of various meta-heuristic algorithms to solving the TSP.

6. ACKNOWLEDGMENTS

Special appreciation goes to Mr. Dominic Asamoah and Mr. Emmanuel O. Oppong of Kwame Nkrumah University of Science and Technology, Computer Science Department.

7. REFERENCES

[1] Applegate, D. L., Bixby, R.E., Chvátal, V., and Cook, W. J. 2006. The Travelling Salesman Problem: A Computational Study. Pinceton Series in Applied Mathematics: Princeton.

[2] Clarke, G. and Wright, J.W. 1964. Scheduling of vehicles from a central depot to a number of delivery points. Oper. Res., vol. 12, pp. 568–581, http://dx.doi.org/10.1287/opre.12.4.568

[3] Whitely, D. Starkweather, T. and D'Ann, F. 1989. Scheduling problems and travelling salesman: The genetic edge recombination operator, in Proc.3rd Int. Conf. Genetic Algorithms, pp.133–140.

[4] Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P. 1983. Optimization by simulated annealing, Science, vol. 220, pp. 498–516, http://dx.doi.org/10.1126/science.220.4598.671

[5] Alizadeh, F., Karp, R. M., Newberg, L. A., and Weisser, D. K. 1993. Physical mapping of chromosomes: A combinatorial problem in molecular biology," in Proc. 4th ACM-SIAM Symp. Discrete Algorithms (SODA), pp. 52–76.

[6] Korostensky C. and Gonnet, G. H. 2000. Using traveling salesman problem algorithms for evolutionary tree construction, Bioinformatics, vol. 16, no. 7, pp. 619–627, http://dx.doi.org/10.1093/bioinformatics/16.7.619

[7] Zelinka, I. 2002. Umělá intelligence v problémech globální optimalizace. Praha: BEN-technická literatúra.

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

[9]http://www.cs.sfu.ca/CourseCentral/125/tjd/tsp_example.html

[10] Eugene Lawler, L., Lenstra, J.K., Rinnooy Kan, A.H.G, and Shmoys, D.B., 1985. The Traveling Salesman Problem, John Wiley & Sons.

[11] http://lcm.csa.iisc.ernet.in/dsa/node187.html.

[12] Dantzig, G. B., Fulkerson, D. R., and Johnson, S. M., 1954. Solution of a large-scale traveling-salesman problem, Operations Res 2: 393–410.

[13] Land, A. H., and Doig A. G., 1960. An automatic method of solving discrete programming problems, Econometrica 28: 497–520.

[14] Little J. D. C., Murty K. G., Sweeney D. W and Karel C., 1963. An algorithm for the traveling salesman problem . Opns Res 11: 972–989.

[15] Martin G. T., 1966. Solving the traveling salesman problem by integer programming. Working Paper, CEIR, New York.

[16] Ellis Horowitz, Sartaz Sahni, and Rajasekaran. 1998. Fundamentals of Computer Algorithms. W.H. Freeman and Company, Indian Edition published by Galgotia Publications, 2000.

[17] Oloruntoyin Sefiu Taiwo et al, 2013. Implementation of Heuristics for Solving Travelling Salesman Problem Using Nearest Neighbor And Nearest Insertion Approaches, International Journal of Advance Research. Volume 1, Issue 3, March 2013, Online: ISSN 2320-9194