Journal of Innovative Research in Engineering and Science 2(2), April, 2011.
ISSN : 2141-8225
© Global Research Publishing, 2011.

A Comparative Evaluation of Selected Heuristic Solutions of Vehicle Routing Problems in Supply Chain Management *(pp 81-93).*

O. O.Okediran, A. E.Okeyinka, O. T.Arulogun, R. A. Ganiyu & ALO, O. O.
Department of Computer Science & Engineering,
Ladoke Akintola University of Technology,
P.M. B. 4000, Ogbomoso, Nigeria
Correspondence e-mail: dotunokediran@yahoo.com

Abstract: This work involves evaluation of three heuristics which are often used to solve the VRP namely; nearest insertion, nearest neighbour, and tour improvement heuristics using computer execution time, implementation complexity and asymptotic time complexity. The research findings of this work have established that the nearest neighbour heuristic is the most efficient in terms of execution time, program volume and programming effort. The nearest insertion heuristics ranks next while the tour improvement heuristic is the least efficient. Solution to a thirty-seven-node vehicle routing problem implemented using the Nigerian State capitals and Abuja was also obtained.

Key words: combinatorial optimization, Impractical problem, Heuristics, Asymptotic time complexity and Halsted complexity measure.

1 INTRODUCTION

The vehicle routing problem (VRP) is often described as the problem in which vehicles based on a certain depot are required to visit geographically dispersed customer in order to fulfill known customer demands. The problem is to construct a low cost, feasible set of routes (one for each vehicle). A route is a sequence of locations that a vehicle must visit along with the indication of the serve it provides. The vehicle must start and finish its tour at the depot.

We can say that the problem arises as a generalization of the travelling salesman problem. The travelling salesman problem (TSP) requires the determination of a minimal cost cycle that passes through each node of a given graph exactly once. If cost are symmetric, that is if the cost of travelling between two locations does not depend on the direction of travel, we have a symmetric TSP, otherwise, we have an asymmetric TSP. The multiple travelling salesman problem arises if many salesmen or vehicles in the fleet are to leave from and return to the same depot. There are no restrictions on the number of nodes that each vehicle must visit except that each vehicle must visit at least one node.

The vehicle routing problem has a variety of additional constraints and extensions that are often found in real world problems they include the following

- Each vehicle can operate on more than one route, provided that the total time spent on these routes is less than a given bound *T*.
- Each customer must be visited within a specific time interval, know as time windows.
- The problem may involve both deliveries to and collections from customers. In addition, it may be possible to mix deliveries and collections on a single route, or alternatively, it may be required from a vehicle to first perform all the deliveries in the route before performing the collection.
- Vehicles may also be associated with time windows within which they are allowed to operate.

The VRP naturally arises as a sub-problem in many transportation and logistics problems, for example the problem of routing of trucks for parcel post pickup, arrangement of school bus routes to pick up the children in a school district, the delivery of meals to homebound persons by fast-food firms and the vehicle routing problem in which vehicles based on a central depot are required to visit geographically dispersed customers in order to fully known customer demands. The problem is to construct a low cost and feasible set of routes for each vehicle. Although transportation applications are the most natural setting for the vehicle routing problem, the simplicity in formation of models for problems modelled after it has led to many interesting applications in other areas. A classic example is the scheduling of a machine to drill holes in a circuit board or other object. In this case the holes to be drilled are the nodes, and the cost of travel is the time it takes to move the drill head from one hole to the next. The technology for drilling varies from one industry to another, but whenever the travel time of the drilling device is a significant portion of the overall manufacturing process then the VRP can play a role in reducing costs.

Furthermore, offshore pipeline design is an area of application of traveling salesman problem. This problem can be described as follows: An oil company owns several oil drilling platforms and oil that is recovered from the platforms must be transported to refineries. A network of pipelines must be constructed between the platforms and the refineries. The concern now is how should the pipeline network be constructed to minimize cost?

Many algorithms have been written to solve these combinatorial problems and the execution time of these algorithms are, however too large and even the computer resources cannot handle such algorithms. Consequently, there is a need to get an alternative means of solving these combinatorial problems. When it is impracticable to compute an optimal solution, there is the need to settle for good but not necessarily optimal solutions. Such solution procedures are called heuristic methods or simply heuristics.

Heuristic is a branch of artificial intelligence (AI) that deals with the building of machines, which provides solution to real life problems. Heuristic algorithms produce a feasible but not necessarily optimal solution **(**Aarts and Stehouwer, 1993). They are generally very fast and efficient means of solving combinatorial problems in which the solution space explodes exponentially (Goodman and Hedetniemi, 1984).

Many heuristics have been developed for solving VRP, but this project is interested in three heuristics namely nearest insertion, nearest neighbour and tour improvement heuristics. The nearest neighbour and nearest insertion heuristics are examples of a broad class of heuristics called tour construction heuristics. Such heuristics build a tour from scratch by a growth process until a feasible tour is constructed, while tour improvement heuristic approach, start from a legal tour and try to improve its quality through neighborhood search (Hoffman and Padberg, 1993). Tour construction heuristics not only serve as plausible mechanisms for generating initial tours needed by local search algorithms, they also provide a unique perspective from a theoretical point of view and good results in practice (Hoffman and Padberg, 1993).

2 RESEARCH METHODOLOGY

- **Asymptotic Analysis**

Asymptotic analysis is based on the idea that as the problem size grows, the complexity can be described as a simple proportionality to some known function. This idea is incorporated in the "Big Oh" notation for asymptotic performance.

Definition: T(n) = O(f(n)) if and only if there are constants c_0 and n_0 such that T(n) $\leq c_0$ f(n**)** for all n $\geq n_0$. The expression ″T(n) = O(f(n))″ is read as "T of n is Big Oh of f of n." The following functions are often encountered in computer science Big Oh analysis:

- T(n) = O(1). This is called constant growth.
- T(n) = O(lg(n)). This is called logarithmic growth.
- T(n) = O(n). This is called linear growth.
- T(n) = O(n log n). This is called "n log n" growth. T(n) grows proportional to n times the base 2 logarithm of n.
- T(n) = O(n^k). This is called polynomial growth.

- $(T(n) = O(2^n))$. This is called exponential growth (Curtis, 1981).

The growth patterns above have been listed in order of increasing "size", that is,

$$O(1), O(\lg(n)), O(n \lg(n)), O(n^2), O(n^3), \dots, O(2^n). \quad (1)$$

- **Halsted Complexity Measures**

Halsted complexity measurement was developed to measure a program module's complexity directly from source code, with emphasis on computational complexity. The measures were developed by late Maurice Halsted as a means of determining a quantitative measure of complexity directly from operators and operands in the module (Halstead, 1977).

Halsted argued that algorithms have measurable characteristics analogous to physical laws. His model is based on four different parameters the number of distinct operators (instruction types, keyboards, e.t.c.) in a program, called n_1; the number of distinct operands (variables and constants), n_2; the total number of occurrences of the operators, N_1; and the total number of occurrences of the operands, N_2. From those four counts, a number of useful measures can be obtained. The number of bits required to specify the program is called the volume V of the program and is obtained through the equation
Program Volume

$$V = N \log_2 n \quad (2)$$

where

$$n = n_1 + n_2 \quad (3)$$

and,

$$N = N_1 + N_2 \quad (4)$$

In an attempt to include the psychological aspects of complexity in the measures, Halsted studies the cognitive processes related to the perception and retention of simple stimuli. Research by Stroud in 1966 has shown that the mean number of mental discriminations per second in average human being, also called the Stroud number, is between 5 and 20. Halsted uses 18 as a reference point for his studies. In his model, the number of discrimination made in the preparation of a program, called effort, is given by
Programming Effort,

$$E = V / L \tag{5}$$

where L is program level defined as,

$$L = (n \ln 2) / n_1N_2 \tag{6}$$

And intelligence content of a program is given by,

$$I = L^* V \tag{7}$$

All of these measures are valid under the assumption that the program is "pure", i.e. free of "poor programming practices". Halsted defined six classes impurities, among them are: synonymous operands, unfactored expressions, and common sub expressions.

- **Problem Formulation**

The tour construction (nearest neighbour and nearest insertion) and tour improvement heuristics were employed in providing an optimal solution to the minimization problem modelled after a travelling salesman who is required to visit once each of thirty-six state capitals in Nigeria and the federal capital, Abuja, before returning home. He knows the distance between each of the state capitals and wishes to minimize the total distance travelled. The problem is; in what order should he visit the cities without taking any route twice before returning to the starting or initial city? Each state capital will denote our nodes and the distances between them, the edges of the network. The heuristics were coded to obtain the shortest possible route from a certain node *A* to all the other thirty six nodes then back to the starting point. A comparative evaluation of the software implementation of the heuristics was then carried out to determine the most efficient among the three heuristics.

- **Nearest Neighbour Heuristic**

The heuristic starts with a tour containing a randomly chosen city and then adds to the last city in the tour, the nearest not yet visited city. The algorithm stops when all cities are on the tour. The heuristic is stated below:
Step 1: Begin with any vertex i and find city j such that d (i,j) is the smallest among all j.
Step 2: Next, find the closest city to j that is not already in the tour, say vertex k, and add edge (j,k) to the tour.
Step 3: Repeat this process until the last vertex is added and then join the first and last vertices by the unique edge between them (Nilsson, 1998).

- **Nearest Insertion Heuristic**

The heuristic starts with a tour consisting of an arbitrary city and then choose in each step a city *k* not yet on the tour, which is nearest to the arbitrary city chosen. The heuristic is stated below:

Step 1: Select one vertex to start with, say vertex i.
Step 2: Choose the nearest vertex, say j and form the sub tour i→j→i
Step 3: At each iteration find the vertex k not in the sub-tour that is closest to any vertex in the sub-tour.
Step 4: Find the edge that minimizes d(i,k) + d(k,j) – d(i,j).
Step 5: Insert vertex in between i and j.
Step 6: Repeat this process until a tour is constructed (Nilsson, 1998).

- **Tour Improvement Heuristic**

The heuristic modifies the current solution by replacing *k* edges in the tour by *k* new edges so as to generate a new improved tour. The improvement heuristic is also known as exchange heuristic (Potvin, 1993). The exchange is applied iteratively until an optimal solution is found; a tour which cannot be improved further via the exchange heuristic. The heuristic is stated below:

Step 1: Delete two non-consecutive edges from the graph H.
Step 2: Reconnect the nodes so that the graph H still forms a tour H'
Step 3: If the new tour H' has a lower weight, then H' replaces H.
Step 4: Repeat the process until there is no further improvement (Nilsson, 1998).

3 SYSTEM IMPLEMENTATION

The nearest neighbour, nearest insertion and tour improvement heuristics were implemented using Visual Basic programming language platform The program has two graphical user interfaces (GUI); the first serves as the input interface while the second is the output interface. The input interface allows users to select one out of the three heuristic at a time, after which the starting or initial city will then be chosen.

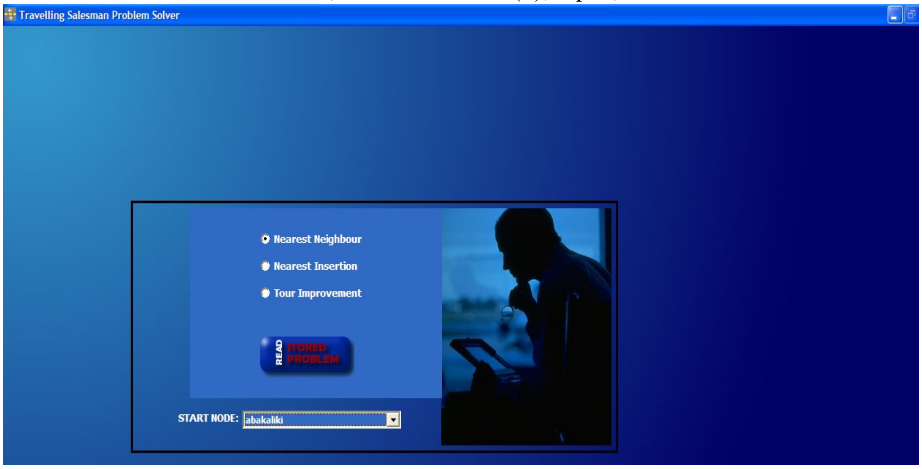
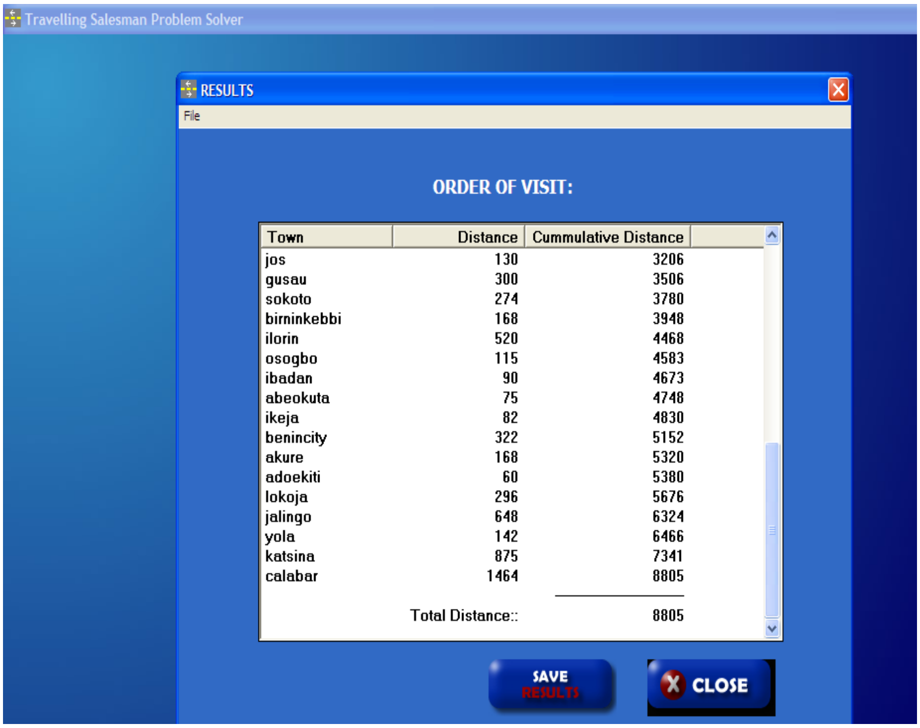


Figure 1: Input Graphical User Interface

The output graphical user interface shows the order of visit of each of the cities, the distances between consecutive cities and the cumulative distance.



ORDER OF VISIT:

Town	Distance	Cummulative Distance
jos	130	3206
gusau	300	3506
sokoto	274	3780
birninkebbi	168	3948
ilorin	520	4468
osogbo	115	4583
ibadan	90	4673
abeokuta	75	4748
ikeja	82	4830
benincity	322	5152
akure	168	5320
adoekiti	60	5380
lokoja	296	5676
jalingo	648	6324
yola	142	6466
katsina	875	7341
calabar	1464	8805
	Total Distance::	8805

Figure 2: Output Graphical User Interface

4 RESULTS AND DISCUSSION

The Visual Basic programming language implementation of the three heuristics was implemented on two computer systems with different configurations: Pentium IV 1.60GHz processor with 512 MB of DDRAM and Pentium III 600MHz processor with 256MB of SDRAM. The table below shows the computer execution time (in millisecond) of each heuristic on the two computer systems.

Table 1: Computer Execution Time

	Nearest Neighbour (ms)	Nearest Insertion (ms)	Tour Improvement (ms)
Pentium IV 1.60GHz, 512 MB DDRAM	20.03	20.07	20.15
Pentium III 600Hz, 256 MB SDRAM	40.07	40.23	50.50

The Halsted metrics computations of the three Heuristics are in appendix B. However, the following tables and view graphs show the values of the counted and calculated metrics of the Heuristics as extracted from the appendix.

Table 2: Counted Metrics of the Three Heuristics

	Number of distinct operators n_1	Number of distinct operands n_2	Total number of occurrence of operators N_1	Total number of occurrence of operands N_2
Nearest Neighbour	4	12	16	42
Nearest Insertion	6	25	31	82
Tour Improvement	5	27	40	107

Table 3: Calculated Metrics of the Three Heuristics

	Program Volume	Programming Effort
Nearest Neighbour	232	3515
Nearest Insertion	560	12821
Tour Improvement	735	17728

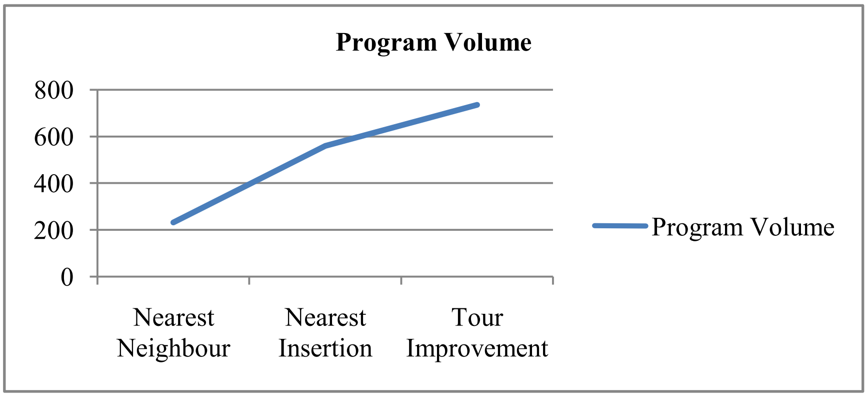


Figure 3: Program Volume

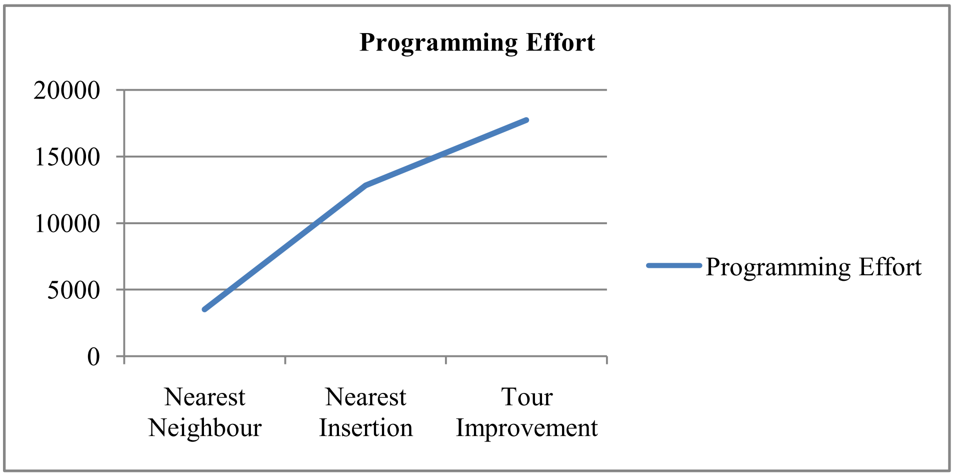


Figure 4: Programming Effort

Table 4 shows the asymptotic time complexity, speed ranking of the heuristics based on their running time and the software complexity measures of the three heuristics studied in this research work.

Table 4: Comparison of Asymptotic Time Complexity, Speed ranking and Software Complexity

	Asymptotic Time Complexity	Speed Ranking	Program Volume	Programming Effort
Nearest Neighbour	$O(n^2)$	1	232	3515
Nearest Insertion	$O(n^2)$	2	560	12821
Tour Improvement	$O(n^2)$	3	735	17728

Measure of Nearest Neighbour, Nearest Insertion and Tour Improvement Heuristics.
The research finding of this work shows that there are differences amongst the software complexities of the three heuristics in terms of the computer execution time, program volume and programming effort.

The nearest neighbour has least computer execution time while the tour improvement heuristic has the highest execution time. Also the nearest neigbour heuristic has the least program volume as well as the least programming effort. Nearest insertion heuristic ranks next both in terms of program volume and programming effort while tour improvement has more volume than both the nearest neighbour and nearest insertion heuristic and also requires more effort to be developed. Generally, programming effort is noted to be directly proportional to program volume. The asymptotic time complexity of the three heuristic runs in $O(n^2)$.

5 CONCLUSION

This work has examined various issues on complexities of three heuristics amongst many that can be employed in solving the travelling salesman problem. The results of the research show that the nearest neighbour heuristic is the most efficient based on the facts that it is the fastest in terms of execution time, least in program volume and does not require much programming effort. On the other hand however, the tour improvement heuristic is the least efficient because of its large execution time, bulky lines of code hence, tedious programming effort.

It is however worthy to note that the asymptotic time complexity of the three heuristics has a polynomial growth running in the value $O(n^2)$ which is true for most combinatorial optimization problem(Johnson and McGeoch, 2002).

6 REFERENCES

Aarts E. H. and Stehouwer H. P. (1993). *Neural Networks and the Travelling Salesman Problem*, Springer Verlag, Berlin, pp 14-17.

Goodman S. E. and Hedetniemi S. T. (1984). *Introduction to the design and analysis of Algorithms*, Second edition, Princeton University Press, Princeton, N.J, pp 91.

Grotschel M. and Padberg M. W. (1985), *Polyhedral Theory: The Traveling Salesman Problem*, Lawler, Lenstra, Rinooy Kan and Shmoys, eds., John Wiley, pp 251-306.

Halsted, M. H. (1977). *Elements of Software Science*, Elsevier North-Holland, New York, pp 45.

Hoffman K. and Padberg M. W., (1993). *Traveling Salesman Problem.* Working Paper 90/1, Department of Computer Science, New York University.

Johnson D. S. and. Mcgeoch L. A. (1995). *The Traveling salesman problem*: A Case study in Local Optimization", *Documenta Mathematica* - Extra Volume, ICM III, pp 645-658**.**

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

Nilsson C., (1998). *Heuristics for the Traveling Salesman Problem* PDRC Report Series No. 81-10. Georgia Institute of Technology, Atlanta, Georgia

APPENDIX

Halsted Metric Computation

Nearest Neighbour Heuristic

The number of distinct operators used in the program is four while the total number of operators' occurrence in the program is sixteen.

$n_1 = 4$

$N_1 = 16$

The number of distinct operands used in the program is twelve while the total number of operands' occurrence in the program is forty two.

$n_2 = 12$

$N_2 = 42$

$N = N_1 + N_2 = 16 + 42 = 58$

$n = n_1 + n_2 = 4 + 12 = 16$

Program Volume $V = N \log_2 n$

$V = N\frac{log n}{log 2}$=58$\frac{log 16}{log 2} = 232$

Program Level L = (n ln 2) / n_1N_2
L = (16 * ln 2)/ (4 x 42) = 0.066

Programming Effort E = V / L
E = 232 / 0.066 = 3515

Nearest Insertion Heuristic
The number of distinct operators used in the program is six while the total number of operators' occurrence in the program is thirty one.
$n_1 = 6$
$N_1 = 31$
The number of distinct operands used in the program is twenty five while the total number of operands' occurrence in the program is eighty two.
$n_2 = 25$
$N_2 = 82$
$N = N_1 + N_2 = 31 + 82 = 113$
$n = n_1 + n_2 = 6 + 25 = 31$

Program Volume V = N $\log_2$ n

$V = N\frac{log n}{log 2}$=113$\frac{log 31}{log 2} = 560$

Program Level L = (n ln 2) / n_1N_2
L = (31 * ln 2) / (6 x 82) = 0.04367

Programming Effort E = V / L
E = 560 / 0.04367= 12821

Tour Improvement Heuristic
The number of distinct operators used in the program is five while the total number of operators' occurrence in the program is forty.
$n_1 = 5$
$N_1 = 40$

The number of distinct operands used in the program is twenty seven while the total number of operands' occurrence in the program is one hundred and seven.

$n_2 = 27$

$N_2 = 107$

$N = N_1 + N_2 = 40 + 107 = 147$

$n = n_1 + n_2 = 5 + 27 = 32$

Program Volume $V = N \log_2 n$

$$V = N\frac{log n}{log 2}{=}147\frac{log 32}{log 2} = 735$$

Program Level $L = (n \ln 2) / n_1N_2$

$L = (32 * \ln 2) / (5 \text{ x } 107) = 0.04146$

Programming Effort $E = V / L$

$E = 735 / 0.04146 = 17728$