Evaluation of Complexity of Some Programming Languages on the Travelling Salesman Problem

D. R. Aremu
O. A. Gbadamosi (Mrs)*

Department of Computer Science
University of Ilorin
Ilorin, Nigeria.

Abstract

The classical Travelling Salesman Problem and its approximate solution by means of the Nearest Neighbour Heuristic were presented in this study. The objective of the work was to find out the performance of some selected programming languages in solving the TSP. Being an NP- complete problem, it was considered that any means or techniques of reducing the time taken in solving the TSP was a welcome development; hence the need for using very efficient programming languages in developing software having many NP-complete modules cannot be over emphasised. In this study the Nearest Neighbour Heuristic was implemented using C++, C#, and Java. The complexities of the three programs using the Halstead complexities measures were computed. Also, the computer execution times of the three programs were captured. A 10-city TSP was used for illustration. Using the computer execution times, the study showed that both C++ and C# are faster than Java in solving the TSP. The complexities of the C++ programs are the highest, followed by those of C# and Java respectively. These results are in agreement with the fact that C++ is a middle- level language, whereas both C# and Java are higher-level languages. The programming implication of this is that a C++ programmer writes codes more from the scratch than a programmer using C# or Java. The study was concerned only with the classical TSP formulation.

Keywords: Complexity, TSP, Halstead Metrics, Computer Execution time

1. Introduction

The Travelling salesman problem [TSP] is one of the classical graph problems which still has no practical exact solution; yet it is a model that can be used to represent many real life problems in routing and scheduling in general. It finds relevance in marketing, irrigation, pipe laying, networking, transportation, communication, and so forth. So far heuristics or approximate algorithms are being used to solve those problems that could be modeled as the TSP. Though very simple to characterize it is notoriously difficult to solve; it belongs to the class NP-complete. It has motivated a number of different approaches, including cutting-plane methods in integer programming, neural nets, simulated annealing and the design of various heuristics [Jonathan L. Gross et.al, 2006]. The TSP naturally arises as a sub-problem in many transportation and logistical 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 person by fast-food firms, and so forth.

2. Motivation and Statement of the Problem

2.1 One of the measures used to evaluate performance of algorithms is the computer time which is a function of programming language, compiler being used, computer hardware and programmer's ability (effectiveness).
In recent times, many programming languages are available but little attention is paid to how they perform in the area of software development. However, the development witnessed by the software industry in the recent years has made it imperative for programmers to use efficient programming languages that are easy to understand and implement. Therefore a proper selection of an appropriate language for implementing algorithm is a step in the right direction. This study evaluates the performance of three popular programming languages namely: Java, C++ and C# using the classical Traveling Salesman Problem (TSP) as the problem domain (Fenwa O. D. et. al, 2012).

2.2. Aim and Objectives: The aim of this study is to evaluate the complexity of three different programming languages as tools of implementation, of the Nearest Neighbour Heuristic in solving a typical travelling salesman problem.

Specially the Objectives are to:

i. To use three similar object oriented programming languages in solving TSP.
ii. To evaluate the performance of the three languages using Halstead Measure.
iii. To evaluate the performance of the three languages using computer execution times.

3. Methodology

The Traveling Salesman Problem (TSP) was studied. Arising from the study a 10-city TSP was modeled and a classical heuristic; the Nearest Neighbour Heuristic, was chosen to solve the TSP. The choice of the Nearest Neighbour Heuristic was informed by the fact that it is the most basic and widely used to illustrate the TSP in literature. The Nearest Neighbour Heuristic was programmed using three popular object-oriented programming languages viz: C++, C# and Java. Halstead metrics which measure program complexity directly from the high level implementation (source code) were used to measure the complexity of these languages. Also, the computer execution times were captured and taken into consideration in evaluating the performance of the three languages.
Halstead complexity measurement was developed to measure program's complexity directly from source code, with emphasis on computational complexity. The metrics were developed by Maurice Halstead as a means of determining a quantitative measure of complexity directly from operators and operands in the program. Halstead argued that algorithms have measurable characteristics analogous to physical laws. His model is based on four different parameters.

1. The number of distinct operators (instruction types, +, -, /, etc); n_1.
2. The number of distinct operands (variables and constants); n_2.
3. The total number of occurrences of the operators; N_1.
4. The total number of occurrences of the operands; N_2

Using these metrics, the following measures can be computed.

1. Program Length (N)
The Program Length (N) is the sum of total number of operators and operands in the program: $N = N_1 + N_2$

2. Vocabulary Size (n)
The Vocabulary Size (n) is the sum of the number of distinct operators and distinct operands: $n = n_1 + n_2$

3. Program Volume (V)
This is the information contents of the program. $V = N \times \log_2(n)$

4. Difficulty Level (D)
The difficulty level of a program is proportional to the number of distinct operators in the program.
$D = (n_1/2) \times (N_2/n_2)$.

4. Implementation

4.1 Modeling TSP as a Graph

A 10-city TSP below was considered and solved using the Nearest Neighbour Heuristic. Three different programming languages were used to implement the Nearest Neighbour method for solving this problem.
A complete enumeration solution would give us $\frac{1}{2}(n-1)!$ Solutions generally and in this case it would have given us $\frac{1}{2}(10-1)!$

$= \frac{1}{2}(9)!$ solutions

$= 181,440$ solutions.

For a 100-city TSP, we have a solution space that would take the computer running at a speed of computing one solution per microsecond 10^{140} centuries (H. P. Williams)

A 10-city TSP: A TSP can be modeled as a graph. Figure 4.1 below is a graph of a 10-city TSP.

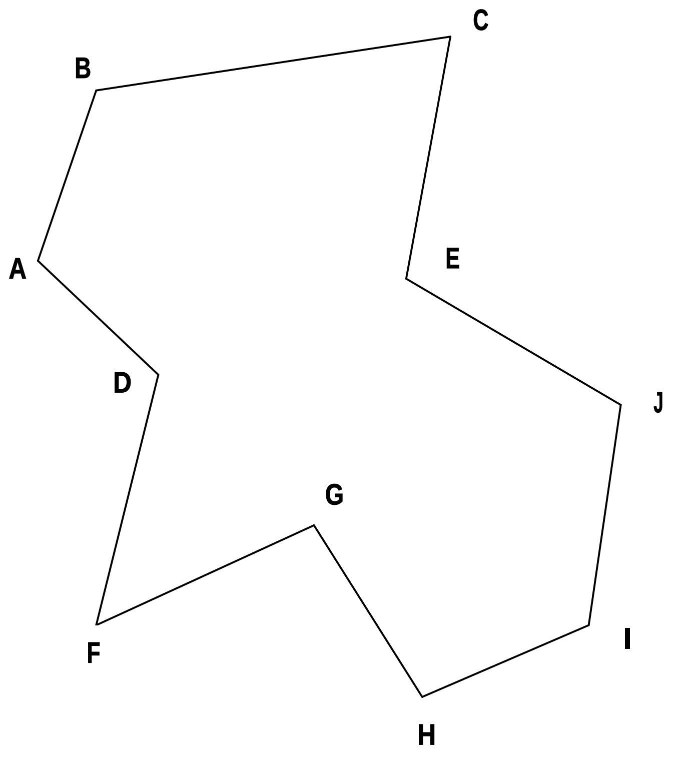

Fig. 4.1: graph of a 10-city TSP

So for pragmatic purposes, getting exact solution is a far cry from practicability, hence the need for making recourse to the use of heuristics.

The Nearest Neighbour Heuristics used in this study is as follows.

1. Randomly select a starting node
2. Add to the last node the closest node until no more node is available.
3. Connect the last node with the first node.
4. Stop.

The following is the matrix table of the distances between the ten cities.

	A	B	C	D	E	F	G	H	I	J
A	∞	75	30	55	70	70	80	11	40	32
B	75	∞	55	30	40	15	34	22	15	13
C	30	55	∞	65	45	55	21	82	34	34
D	55	30	65	∞	15	120	67	13	22	15
E	70	40	45	15	∞	20	10	51	55	40
F	70	15	55	10	20	∞	12	98	30	21
G	80	34	21	67	10	12	∞	36	75	82
H	11	22	82	13	51	98	36	∞	67	22
I	40	15	34	22	55	30	75	67	∞	20
J	22	13	34	15	40	21	82	22	20	∞

Table 4.1

4.2 Halstead Complexity Measurements

The number of operands and operators present in an algorithm or a computer program are very basic in the bid to calculate the complexity of such a tool. An operand in computer science is the term used to describe any object that is capable of being manipulated. For example, in “1 + 2” the “1” and “2” are operands and the plus symbol is the operator.

In addition to digits or letters, operands can also be represented by expressions. A combination of letters, numbers or symbols used to represent a value of a variable is called expression. Expressions are found and used in various programming languages, databases and spreadsheets. Examples are:

Sum = zero
Minidex = 0
Range = largest – smallest
Length = matrix.lenght
Matrixcol = newint

In all the cases above, expressions are the terms to the left and right of the equality symbol. Equality symbol need not always be the operator between expressions. Indeed any operator, formal or informal, can be an operator, operating between expressions. In general operands can be operated upon by more than just one operator. For example: In a + b – c; a, b and c are operands while + and – are operators. In range = largest – smallest, range, largest and smallest are operands while = and – are operators.

In this study, the following were used as operators

=, +, -, >, <, ==,&&, i++, j++, /

In computing the complexity, Halstead measures were used. Halstead measures estimate complexity metrics directly from the operators and operands in the source code. The measurable and countable properties are:

• n_1 = number of unique or distinct operators appearing in the code
• n_2 = number of unique or distinct operands appearing in the code
• N_1 = number of all the operators appearing in the code
• N_2 = total number of all the operands appearing in the code.

From these metrics, Halstead defines

• The vocabulary (n) as $n = n_1 + n_2$
• The program length (N) as $N = N_1 + N_2$

The following complexities were then calculated

(i) Program length (N) = $N_1 + N_2$
(ii) Vocabulary size (n) = $n_1 + n_2$
(iii) Volume (V) = $N \times \log_2 (n)$
(iv) Difficulty level (D) = $(n_1/2) \times (N_2/n_2)$

4.3 Overall Control Structure

The following algorithm depicts the overall structure of the three programs.

```
Begin
    Initialise All Variables
    Supply Input Data
    Begin
        Repeat
            Read input data
        Until all data are read
    End;
    Start with an arbitrary node
    Begin
        Repeat
            Visit the Nearest Node recursively
        Until all nodes have been visited
    End;
    Join the last node with the first node
End
```

5. Analysis of Results

The results obtained from our implementation detailed in section (4) are presented and discussed in this section.

5.1 Computing the Complexities

In computing the complexities, the declaration part, the input part as well as the output part of the three programs are not taken into consideration. This is informed by the fact that these three parts constitute basically the main housekeeping functions in running any program on the computer. It is considered that the major differences in the complexities of the program would be implicit in the actual computation peculiar to each program. The following metrics are derived by counting n_1, n_2, N_1, N_2 and manipulating them with respect to each of the three programs.

5.1.1 Analysis of the C++ program
(a) From int arrayMin (int arr [ ]) module:
number of distinct operators $n_1 = 6$
number of distinct operands $n_2 = 10$
number of all operators $N_1 = 13$
number of all operands $N_2 = 25$

(b) From int main ( ) module:
number of distinct operators $n_1 = 3$

number of distinct operands $n_2 = 10$
number of all operators $N_1 = 8$
number of all operands $N_2 = 14$
(c) From the entire program
number of distinct operators $n_1 = 6 + 3 = 9$
number of distinct operands $n_2 = 10 + 10 = 20$
number of all operators $N_1 = 13 + 8 = 21$
number of all operands $N_2 = 25 + 14 = 39$
Hence
(i) Program length (N) = $N_1 + N_2 = 21 + 39 = 60$
(ii) Vocabulary size (n) = $n_1 + n_2 = 9 + 20 = 29$
(iii) Volume (V) = $60\log_2 29$
(iv) Difficulty level (D) = $(n_1/2) \times (N_2/n_2)$
$= (9/2) \times 39/20$
$= 8.775$

5.1.2 Analysis of C# program
number of distinct operators $n_1 = 6$
number of distinct operands $n_2 = 13$
number of all operators $N_1 = 21$
number of all operands $N_2 = 37$
Hence
(i) Program length (N) = $N_1 + N_2 = 21 + 37 = 58$
(ii) Vocabulary size (n) = $n_1 + n_2 = 6 + 13 = 19$
(iii) Volume (V) = $N\log_2 n = 58\log_2 19$
(iv) Difficulty level (D) = $(n_1/2) \times (N_2/n_2)$
$= (6/2) \times 37/13$
$= 8.538$

5.1.3 Analysis of Java program
number of distinct operators $n_1 = 7$
number of distinct operands $n_2 = 12$
number of all operators $N_1 = 21$
number of all operands $N_2 = 35$
Hence
(i) Program length (N) = $N_1 + N_2 = 21 + 35 = 56$
(ii) Vocabulary size (n) = $n_1 + n_2 = 7 + 12 = 19$
(iii) Volume (V) = $N\log_2 n = 56\log_2 19$
(iv) Difficulty level (D) = $(n_1/2)\mathrm{x}(N_2/n_2)$
$= (7/2) \times 35/12$
$= 10.208$

5.2 Results and Discussion

Table 4.6: Results: complexities and computer execution times.

	Program length N	Vocabulary size N	Volume V	Difficulty level D	Computer execution time (nano sec.)
C++	60	29	$60\log_2 29$	8.775	0.0000
C#	58	19	$58\log_2 19$	8.538	0.0000
Java	56	19	$56\log_2 19$	10.208	1337

From our results above, we have the following order:
Program length N: Java < C# < C++

Vocabulary V: (Java = C#) < C++
Program Volume V: Java < C# < C++
Difficulty level D: C# < C++ < Java

Computer Execution Times in Nanoseconds: (C++ = C#) < Java
First and foremost we must understand the basic differences between C++, C# and Java. Such an understanding will inform our opinion about the behaviour of these languages vis-a-vis the programs we analyzed.
First, Java is a general-purpose, concurrent, class-based,

Object-oriented computer programming language that is specifically designed to have as few implementation dependences as possible. It is intended to let application developers, 'write once, run anywhere' (WORA) meaning that code that runs on one platform does not need to be recompiled to run on another. Java is one of the most popular programming languages in use, particularly for client – server web applications. However, the following criticisms have been directed at Java. Its speed of executing program is slower compared to that of other object-oriented programming languages. Also, it has a history of security vulnerabilities in the primary java virtual machine implementation.

C# on the other hand aims to combine the high productivity of Visual Basic and the power of C++. It was designed specifically to work with the .Net and is geared to the modern environment of windows and mouse-controlled user interface, networks and the internet. C++ however was designed to be a mid-level platform-neutral object-oriented programming language. A C++ programmer would do more coding form the scratch than a programmer in C# or Java.

i. From the foregoing therefore, it should not be surprising that both C++ and C# performed better in terms of computer execution times. The time taken by both C++ and C# could not even be measured in nanoseconds; this explains why their computer execution times were zero. Whereas the time taken by Java was 1337 nanoseconds. This confirmed the speed weakness already levied against Java.
ii. The complexities of the C++ program in terms of program length, program vocabulary and program volume are more than those recorded for C# and Java. This also confirms the claim that whereas both C# and Java are higher-level language, C++ is a mid-level language which naturally requires more effort in coding.
iii. In terms of program difficulty, both C++ and C# behave approximately alike with complexities of 8.775 and 8.538 respectively, while Java has a difficulty level of 10.208. An objective and reliable meaning of these metrics is that both C++ and C# could be classified as requiring roughly the same programmers' effort while Java requires a different effort.
These results are not sufficient to conclude that Java is more difficult than C++ and C# and vice-versa. More factors will have to be considered in arriving at such a conclusion.
iv. Finally, considering the fact that the Traveling Salesman Problem is an NP-complete problem, any device or effort that could reduce the time required in solving the problem should be embraced. From our results, it is very clear that though Java has become the most popular programming language, it should not be used in designing software for solving NP-complete problems. As shown by the results in this study, C++ and C# are preferred when the speed of computation is of paramount importance.

6. Conclusion

This study examines the performance of three programming languages viz: C++, C#, and Java in solving the travelling salesman problem using the nearest neighbour heuristic. In addition it also evaluates the performance of these languages using computer execution time. The complexities of the programs written in these languages were computed using the Halstead metrics: Program Length, Vocabulary Size, Program Volume, and Difficulty Level.
Our results showed that both C++ and C# are faster than Java in solving the TSP; also the complexities of C++ are highest. The programming implication of this is that a C++ programmer will have to write codes more from the scratch than a programmer using C# and Java. This is in line with the fact that C++ is a middle-level language, whereas both C# and Java are higher-level languages.

Furthermore, both C++ and C# took approximately the same computer time to execute and get the output, Java took by far higher computer time, that is, Java is slower in speed than both C++ and C#. This also corroborates the known characteristic of Java as a relatively slow language. One of the limitations of this study is that the TSP considered in this work was solved statically; another limitation is that this study assumed that it would always be possible for our hypothetical salesman to be moving to the nearest nodes. Apparently, it may not be possible in real life due to some unforeseen constraints to move always to the nearest nodes.

Hence, while recommending that for a future research the TSP could be solved dynamically to make room for flexibility in the number of cities that this framework could accommodate, it is also recommended that situations whereby it becomes impossible to move to the nearest nodes, should be another candidate that can be considered in future research.

References

Fenwa O.D; Okeyinka A. E; Olabiyisi S.O, and Ajala F.A. (2012); Performance Evaluation of Implementation Languages on the Complexity of Heapsort Algorithm; International Journal of Computer Application; Issue 2 Volume 5, October 2012.

H.P. Williams "The Traveling Salesman Problem", London School of Economics.

National Open University of Nigeria (2006); Lecture Notes on Object- Oriented Programming.

Okediran O. O, 2008. A performance Evaluation of Selected Heuristics to Travelling Salesman Problem.

S.E. Goodman and S.T. Hedetniemi (1997); Introduction to the Design and Analysis of Algorithms; McGraw – Hill Ltd.