

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/264286830>

Implementation and Analysis of Depth-First Search (DFS) Algorithm for Finding The Longest Path

Conference Paper · August 2011

DOI: 10.13140/2.1.2878.2721

CITATIONS

5

READS

10,119

3 authors, including:



Sheila Eka Putri

University of Sumatera Utara

3 PUBLICATIONS 5 CITATIONS

SEE PROFILE



Tulus Tulus

University of Sumatera Utara

152 PUBLICATIONS 632 CITATIONS

SEE PROFILE

Implementation and Analysis of Depth-First Search (DFS) Algorithm for Finding The Longest Path

Sheila Eka Putri¹, Tulus², Normalina Napitupulu³

¹ Department of Mathematics FMIPA USU
putri.sheilaeka@gmail.com

² Department of Mathematics FMIPA USU
tulus@ac.id

³ Department of Mathematics FMIPA USU

Abstract. Depth-First Search (DFS) is one of searching algorithm using data structure Stack when it reaches a node or vertex which connected in a graph. The ability of DFS to find each nodes that have not been visited simplify the searching of optimum solution in some problems include the longest path problem. In this paper we describe implementation of a simple program and analyze the use of algorithm DFS by Java programming (used NetBeans 6.9) which aims to obtain the optimum solution of the longest path problem in a graph. Use a graph representation by adjacency matrix, then the number n in graph is $n = |V(G)|$ the length representation of G . So, there are $n!$ permutation are possible in vertex and therefore the algorithm running time is $\Omega(n!)$. The result obtained an optimum solution of the longest path problem in a graph which can also be used in solving Hamiltonian cycle and Traveling Salesman Problem (TSP) and proved that *Hamiltonian cycle* \leq *Longest Path*.

Key words: longest path, Depth-First Search (DFS), Hamiltonian cycle.

1. Introduction

An algorithm to find k longest paths of a directed acyclic graph (DAG) was presented earlier by Yen et. al [1]. Kundu presented an incremental algorithm to identify longest path by divided forward traversal of nodes which the memory grabbed is order of $O(K*N)$, where N is the number of nodes in a graph and K is selected to be a small number [2].

In this paper we present an implementation and solutions in determining the longest path in a graph based on the length of the graph. Longest path is one of NP problem where there are

no polynomial time to solve it, because longest path has running time complexity $\Omega(n!)$. There are two search algorithms in a graph, Depth-First Search (DFS) algorithm and Breadth-First Search (BFS) algorithm. We use DFS algorithm as our search algorithm that uses data structure *Stack* and process of backtracking in order to determine each vertex that has not been visited 'deeply'. DFS algorithm serves to identify each vertex that has not been visited and took the value of each vertex into the *Stack* which would then be processed to obtain the exact solution or a suboptimal solution. Thus, we used acquisition method of optimum solution, namely dynamic programming, approximation and heuristic as a method in obtaining exact solutions in determining the longest path in a graph.

Implementation of DFS algorithm will identify a search on each path to determine the longest path in a graph and then the exact solution or a suboptimal solution will be determined by acquisition methods of optimum solution, namely dynamic programming, approximation and heuristic. This application also has been stated previously by Schmidt et. al, in a worst-case calculation of delay in an Ethernet packet to be the solution to the issue of the longest path of a weighted, directed graph. [3]

2. Depth-First Search (DFS) Algorithm

Searching or tracking is one method to solve general problems, especially AI (Artificial Intelligence) problems. Searching is a process of looking for a solution of a problem through a set of possible state space (state space). Depth-First Search (DFS) algorithm is a method that included of blind search. That is, the search is done by ensuring all vertices have been visited, but without the exact solution or a suboptimal solution. [4]

According to Cormen, et al (1992), the main factor which is owned by the DFS algorithm is its ability to find the nodes or vertices that have not visited in depth, and data structure *Stack* that served to 'remember' or store a value in the algorithm reaches a vertex particular.

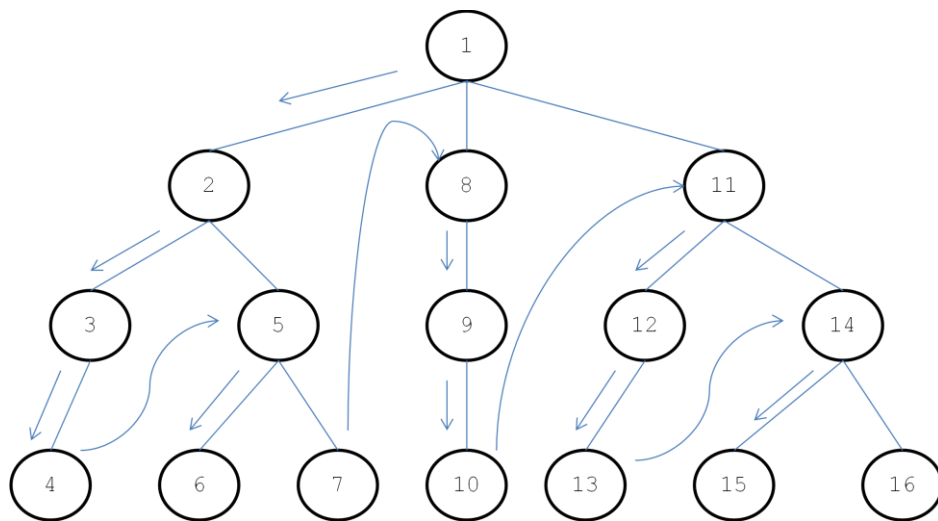


Fig. 1 Search method of DFS algorithm

The pseudo-code of DFS algorithm can be written as follows:

```
dfs-visit(G,v){
    Stack s;
    s.push(start);
    while(s.empty() == false){
        top = s.stop();
        s.pop();
        mark top as visited;
    }
}
```

Longest path problem can be efficiently solved by DFS algorithm.

3. Acquisition Method of Optimum Solution

As described in Section 2, the DFS algorithm is a search method that includes of blind search. So, its ability is just to make sure all the vertices have been visited, but without the exact solution or a suboptimal solution. Then store the value of each vertex using data structure `Stack`.

To obtain the exact solution or suboptimal solutions, DFS algorithms require acquisition methods in guiding the optimum solution algorithm to obtain suboptimal solutions. Than it became an optimum solution of the longest path problem exactly. Here we use dynamic programming, approximation and heuristic as an acquisition method in obtaining the optimum solution as an exact solution for the longest path problem.

a. Dynamic programming

Cormen, et al (1992) explains, dynamic programming is an important role in optimization problems that have a value that will be taken to be the optimum solution either minimum or maximum value. [1]

The development of dynamic programming method is divided into four, namely:

1. Characterization of the structure at an optimum solution.
2. Successively set at an optimum solution value.
3. Calculating the value of an optimum solution in a bottom-up.
4. To construct an optimum solution based on the calculation.

b. Approximation

There are two types of approaches in the NP-Complete problem. First, if input is small, an algorithm with execution time (running time) exponential would produce good output. Second, there is a possibility of obtaining an optimum solutions are approached in

polynomial time. A method that can be used in achieving an optimum value by the closest value is approximation. This method has limitations relative error ϵ . [1]

c. Heuristic

Heuristic search is a method that develops efficiency in the search process, but with the possible expense of completeness (completeness). Heuristic function used to evaluate the circumstances of individual problems and determine how far it can be used to obtain the desired solution. Heuristic information is just a guide to guess the next step and is often done based on experiments or experiments in intuition.

4. Longest Path Problem

Optimization problem is a problem that requires finding the optimum solution, both maximization or minimization. The issue is one of the longest trajectory optimization problem maximizing sample. The problem is also the longest trajectories used in the scheduling of routes and drilling machine to drill holes on a *PCB* with a vertex as part of the machine to be drilled and the edge as the time needed to perform retooling.

Longest path problems is a problem where there is no polynomial time solution to resolve it, which has complexity $\Omega(n!)$. In other words, longest path is one of NP problems. Thus, the equation is obtained that there is implementation of the longest path problem is a Hamiltonian Cycle and Traveling Salesman Problem (TSP) which is also the complexity of $\Omega(n!)$, because the principle whereby all vertices visited exactly once.

Theorem 1. Longest path runs in $\Omega(n!)$.

Proof. Given an instance graph G , one of the decision algorithm which may attach all permutations of the vertices G and then examine the results of all the permutations to see if included into a longest path. We use a graph representation of the adjacency matrix, the number n at the vertices in the graph where $n = |G|$ is the length of representation of G . So, there are $n!$ permutations are possible in the vertices and therefore the running time complexity is $\Omega(n!)$ and proved that longest path is one of NP problem.

5. Experimental and Result

The design software is developed by using one of programming language, Java Programming Language because Java is an open source. Kind of graph which implemented in this program is undirected weighted graph. DFS algorithm was applied to the search problem of longest path. The algorithm can be written as follows:

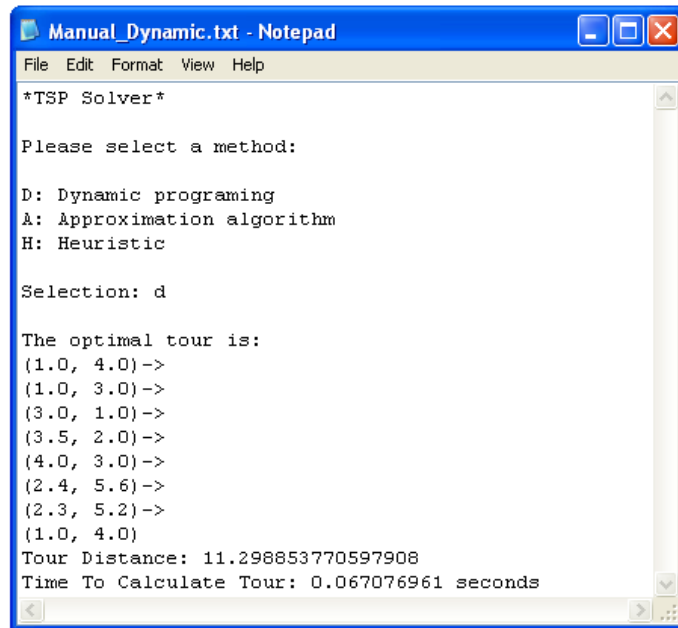
```

public cityXY(double xCord, double yCord) {
    x = xCord;
    y = yCord;
    distance = 0;
    isVisited = false;
    isPrevisitUsed = false;
}
public double getX() {
    return x;
}
public double getY() {
    return y;
}
public void setDistance(double value) {
    distance = value;
}
public double getDistance() {
    return distance;
}
public boolean equals (cityXY checkCity) {
    if(checkCity.getX() == x && checkCity.getY() == y)
        return true;
    return false;
}

```

Boolean value is needed in applying the DFS algorithm in the design of this software. *setPrevisitUsed ()* would be true if vertex not yet visited, and otherwise is false if the vertex has been visited. The role of Boolean values is to guide the DFS algorithm in determining the values of vertices which are to be stored in *Stack*. Suppose there are 7 points which is represented as a city with the coordinates of each point are (1.0, 3.0), (1.0, 4.0), (3.0, 1.0), (2.3, 5.2), (2.4 , 5.6), (3.5, 2.0) and (4.0, 3.0).

By using $distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, whose value is taken by the program with command *getDistance ()*. Thus, the calculation of these solutions is the longest path problem



```
*TSP Solver*

Please select a method:

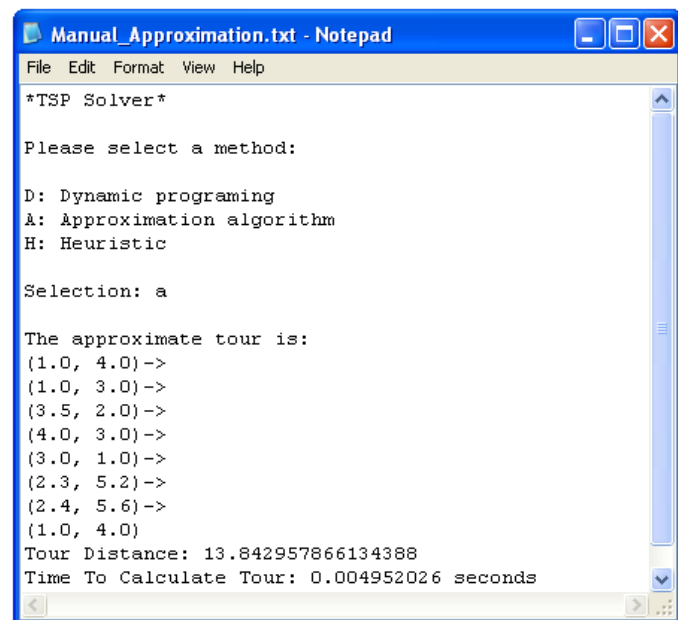
D: Dynamic programming
A: Approximation algorithm
H: Heuristic

Selection: d

The optimal tour is:
(1.0, 4.0)->
(1.0, 3.0)->
(3.0, 1.0)->
(3.5, 2.0)->
(4.0, 3.0)->
(2.4, 5.6)->
(2.3, 5.2)->
(1.0, 4.0)
Tour Distance: 11.298853770597908
Time To Calculate Tour: 0.067076961 seconds
```

Fig. 2 Output by input data Manual and Dynamic method

The output obtained of the longest path is $11.298853770597908 \approx 11.3$ path length. As for the necessary programs running time is $0.067076961 \approx 0.06$ seconds.



```
*TSP Solver*

Please select a method:

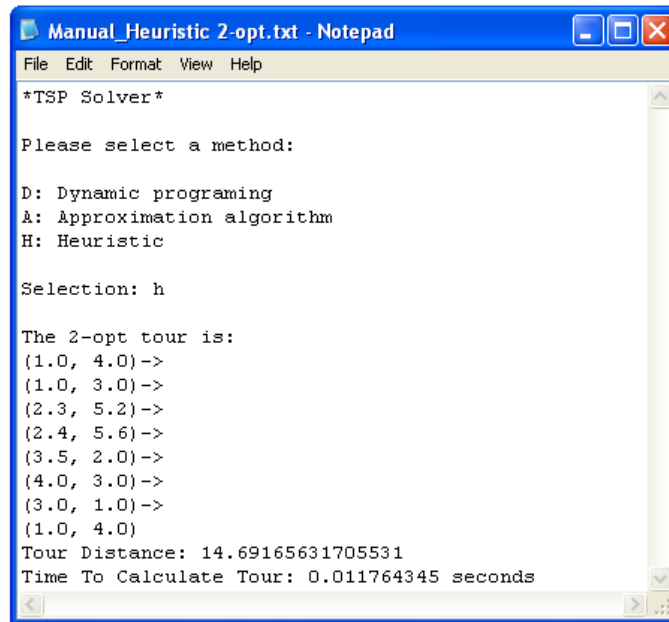
D: Dynamic programming
A: Approximation algorithm
H: Heuristic

Selection: a

The approximate tour is:
(1.0, 4.0)->
(1.0, 3.0)->
(3.5, 2.0)->
(4.0, 3.0)->
(3.0, 1.0)->
(2.3, 5.2)->
(2.4, 5.6)->
(1.0, 4.0)
Tour Distance: 13.842957866134388
Time To Calculate Tour: 0.004952026 seconds
```

Fig. 3 Output by input data Manual and Approximation method

The output obtained of the longest path is $13.842957866134388 \approx 13.84$ path length. As for the necessary programs running time is $0.004952026 \approx 0.005$ seconds.



```
*TSP Solver*

Please select a method:

D: Dynamic programming
A: Approximation algorithm
H: Heuristic

Selection: h

The 2-opt tour is:
(1.0, 4.0)->
(1.0, 3.0)->
(2.3, 5.2)->
(2.4, 5.6)->
(3.5, 2.0)->
(4.0, 3.0)->
(3.0, 1.0)->
(1.0, 4.0)
Tour Distance: 14.69165631705531
Time To Calculate Tour: 0.011764345 seconds
```

Fig. 4 Output by input data Manual and Heuristic method

The output obtained of the longest path is $14.69165631705531 \approx 14.69$ path length. As for the necessary programs running time is $0.011764345 \approx 0.011$ seconds.

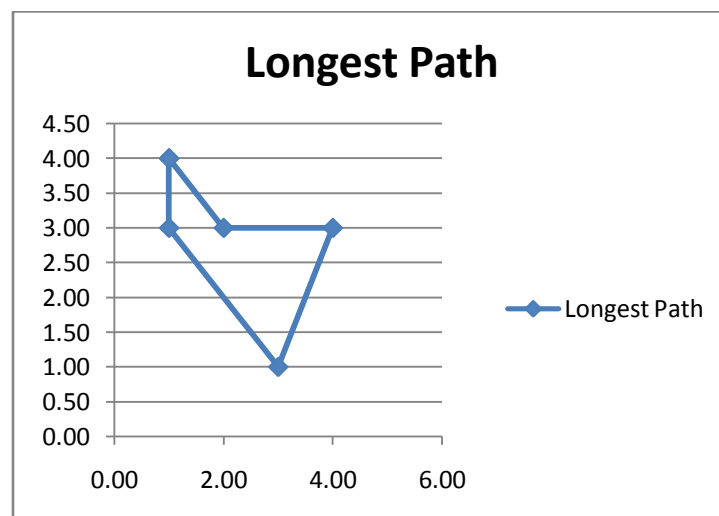


Fig. 5 Longest Path On Test Results

Based on the program result, each acquisition method in obtaining the optimum solution (dynamic, approximation and heuristic) obtained different results.

6. Conclusions

Software is implemented in Java. Our goal is to show that the algorithm DFS algorithm can be one that can be used in the search for the longest trajectory effectively. Hamiltonian cycle is a cycle in an undirected graph which visits each vertex exactly once and also returning to the starting vertex. This is also used in longest path, because each vertex in an undirected graph visits exactly once and obtained the largest path length as the result of the program. Thus, it can be concluded that *Hamiltonian cycle* \leq *Longest path*.

Based on these assumptions, we get the conclusion that the DFS algorithm is an algorithm that is good enough for searching an optimum solution of the longest path problem and prove that the existence of similarities between the longest path and Hamiltonian cycle, in order to obtain a *Hamiltonian cycle* \leq *Longest path*. From the use of all acquisition method of optimum solution, dynamic programming method which implemented in DFS algorithm obtained more accurate and faster running time than two other methods, approximation and heuristic. These methods has different error rates and less work well in accordance with the 'backtracking' principle of DFS algorithm.

Table 1 Data Calculation of Distance Based on Number of Cities

Number of Cities	Tour Distance			Percent Error	
	Dynamic	Approximation	Heuristic	Approximation	Heuristic
1	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.662420	0.662420	0.662420	0.000000	0.000000
3	1.189560	1.189560	1.189560	0.000000	0.000000
4	2.425201	2.425201	2.425201	0.000000	0.000000
5	1.552724	1.635593	1.552724	5.337009	0.000000
6	2.042511	2.042511	2.042511	0.000000	0.000000
7	2.650482	2.843224	2.650482	7.271939	0.000000
8	2.557956	3.151951	2.557956	23.221504	0.000000
9	2.773451	2.929706	2.773451	5.633933	0.000000
10	3.216230	3.216230	3.216230	0.000000	0.000000
11	2.763376	2.974165	3.524637	7.627977	27.548221
12	3.622481	4.591504	3.622481	26.750254	0.000000

Table 2 Data Running Time of Program

Dynamic Programming		Approximation		Heuristic	
Number of cities	Time to solve (sec)	Number of cities	Time to solve (sec)	Number of cities	Time to solve (sec)
1	4.11E-05	1	1.21E-04	1	1.49E-05
2	7.58E-05	2	6.28E-04	2	3.03E-05
3	9.20E-05	3	1.88E-04	3	5.37E-05
4	1.61E-05	4	2.64E-04	4	6.05E-05
5	3.73E-05	5	5.95E-04	5	8.79E-05
6	1.47E-04	6	5.65E-04	6	2.22E-04
7	8.43E-04	7	7.15E-04	7	4.03E-04
8	0.006895	8	6.73E-04	8	7.44E-04
9	0.057800	9	7.14E-04	9	0.001402
10	0.477627	10	0.001061	10	0.002072
11	4.804672	11	0.001129	11	0.002357
12	53.488330	12	0.001135	12	0.004903
...	To long	...	To long	...	To long

References

1. S.H Yen, D.H. Du and S. Ghanta, Efficient algorithms for extracting the K most critical paths in timing analysis, 26th *ACM/IEEE Design Automation Conf.*, July 1989, pp. 649- 654.
2. Klaus Schmidt and Ece G. Schmidt, A Longest-Path Problem for Evaluating the Worst- Case Packet Delay of Switched Ethernet, Department of Electronic and Communication Engineering, Cankaya University, Ankara, Turkey.
3. Cormen, Thomas H., Leiserson, Charles E., and Rivest, Ronald L. 1992. *Introductions to Algorithms*. Cambridge, Massachussets, USA: McGraw-Hill Companies, Inc.
4. Desiani, Anita and Arhami, Muhammad. 2006. *Konsep Kecerdasan Buatan*. Yogyakarta, Indonesia: C.V Andi Offset.
5. http://users.csc.tntech.edu/~algviz/longest_path/algorithms.php. Accessed on Jan 6, 2011, 5:49 AM.
6. <http://trykhoek.weebly.com/traveling-salesman-problem.html>. Accessed on Jan 6, 2011, 4:23 AM.