ABSTRACT

The Traveling Salesman Problem (TSP) is a popular problem, but until now there is no algorithm that has the same search results as brute force with a fast search time. Many algorithms have been made previously related to solving this problem with the aim of finding the shortest route through a number of nodes to finally return to the initial node. The purpose of this research is to create an algorithm that can optimize the search for the shortest route with a fast search time. The approach taken is to find the average value of the distance matrix and look for routes with links that have values below the average value. Each route that has been passed will be marked and compared so that it can facilitate the search with a shorter processing time. In this paper the best and effective routes are limited to 12 nodes. The results obtained show that the Average Score Algorithm provides a relatively stable processing time from node 4 to node 12. The proposed algorithm has a tendency of decreasing processing capacity with increasing number of nodes.

Key word: travelling salesman problem, distance matrix, computation time, complexity, shortest route

ABSTRAK


Kata Kunci: travelling salesman problem, distance matrix, computation time, complexity, shortest route
1. Introduction

Traveling Salesman Problem (TSP) is a classic problem in finding the best route but it is still difficult to be solved conventionally. Problems in the TSP arise if the route sought consists of many nodes, causing a high combination. The high combination causes the computational time needed to find the best route is also high (Abdulkarim, 2015).

The route taken by the TSP can only go through one node once and then go to another node until finally returning to the initial node. This route is also known as the Hamiltonian Cycle. The most common solution for TSP is through brute force by calculating all the shortest route probabilities with the lowest total cost. Calculations using brute force are still possible if the number of n nodes is still below 10 so that the required computing time is still not too high. The number of possible routes chosen using n! and each route will be compared between one another to find the best route. The number of combinations significantly influences the computational time required (Baidoo, 2016).

As a calculation, the number of combinations for 12 nodes can produce as many as 1,99584x107 possible routes. Because TSP is one-way, the combination of route A to node B is the same as B to node A or known as symmetric value, so that the same combination of routes can be eliminated and simplified using the formula (n-1)/2. The higher the number of nodes, the more routes will be compared to each other (Beardwood, 1959).

The large number of nodes makes brute force not the right solution to solve a TSP problem. Therefore we need an algorithm that can be an alternative solution with faster computing time. Although this TSP is a classic problem, but until now there has not been one algorithm that is proven to be able to find the lowest total cost with a short computing time. Research on TSP is not final and continues to the present (Droste, 2017).

2. Related Work

Brute Force algorithm in principle is the flow of problem solving by trying all possibilities that exist to find the best solution. The brute force solution only calculates the total distance for each possible route and then chooses the shortest one. This algorithm was chosen to solve a simple problem and does not require large calculations (Johnson, 2002).

For a large number of nodes, a smarter algorithm is needed to find a shortcut in finding the best solution on the TSP. Intelligent algorithms can eliminate many routes so that computing time is shorter.

Many algorithms have been created as a solution to this problem including the Greedy, Ant Colony Optimization (ACO), Fast Bee Colony Optimization, and others. All of these algorithms try to find the best solution both in terms of total cost and computing time (Girsang, 2012) (Syambas, 2017).

3. Proposed Algorithm

A. Average Value of Distance Matrix

The proposed algorithm in principle looks for the lowest total cost with a short computational time. The first step taken is to calculate the average value of the distance matrix. Distance matrix used in this study is a matrix that has been used by previous studies (Hannah Bast, 2015) (Johnson, 2002).

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>930</td>
<td>993</td>
<td>1138</td>
<td>218</td>
<td>458</td>
<td>1194</td>
<td>637</td>
<td>234</td>
<td>577</td>
<td>529</td>
<td>371</td>
</tr>
<tr>
<td>2</td>
<td>930</td>
<td>0</td>
<td>1748</td>
<td>857</td>
<td>510</td>
<td>901</td>
<td>1022</td>
<td>785</td>
<td>659</td>
<td>421</td>
<td>130</td>
<td>399</td>
</tr>
<tr>
<td>3</td>
<td>993</td>
<td>1748</td>
<td>0</td>
<td>795</td>
<td>795</td>
<td>901</td>
<td>1076</td>
<td>749</td>
<td>795</td>
<td>612</td>
<td>441</td>
<td>609</td>
</tr>
<tr>
<td>4</td>
<td>1138</td>
<td>857</td>
<td>564</td>
<td>0</td>
<td>768</td>
<td>1460</td>
<td>481</td>
<td>930</td>
<td>49</td>
<td>948</td>
<td>381</td>
<td>806</td>
</tr>
<tr>
<td>5</td>
<td>218</td>
<td>510</td>
<td>795</td>
<td>768</td>
<td>0</td>
<td>1380</td>
<td>583</td>
<td>534</td>
<td>434</td>
<td>516</td>
<td>1133</td>
<td>949</td>
</tr>
<tr>
<td>6</td>
<td>458</td>
<td>901</td>
<td>901</td>
<td>1460</td>
<td>1380</td>
<td>0</td>
<td>946</td>
<td>552</td>
<td>549</td>
<td>658</td>
<td>1030</td>
<td>929</td>
</tr>
<tr>
<td>7</td>
<td>1194</td>
<td>1022</td>
<td>1076</td>
<td>481</td>
<td>583</td>
<td>946</td>
<td>0</td>
<td>754</td>
<td>1372</td>
<td>928</td>
<td>836</td>
<td>930</td>
</tr>
<tr>
<td>8</td>
<td>637</td>
<td>785</td>
<td>749</td>
<td>930</td>
<td>534</td>
<td>552</td>
<td>784</td>
<td>0</td>
<td>1257</td>
<td>856</td>
<td>769</td>
<td>616</td>
</tr>
<tr>
<td>9</td>
<td>234</td>
<td>659</td>
<td>795</td>
<td>49</td>
<td>434</td>
<td>549</td>
<td>1372</td>
<td>1257</td>
<td>0</td>
<td>390</td>
<td>757</td>
<td>321</td>
</tr>
<tr>
<td>10</td>
<td>577</td>
<td>421</td>
<td>612</td>
<td>948</td>
<td>516</td>
<td>658</td>
<td>928</td>
<td>856</td>
<td>390</td>
<td>0</td>
<td>681</td>
<td>363</td>
</tr>
<tr>
<td>11</td>
<td>529</td>
<td>130</td>
<td>441</td>
<td>381</td>
<td>1133</td>
<td>1030</td>
<td>816</td>
<td>769</td>
<td>757</td>
<td>681</td>
<td>0</td>
<td>928</td>
</tr>
<tr>
<td>12</td>
<td>371</td>
<td>399</td>
<td>669</td>
<td>806</td>
<td>949</td>
<td>928</td>
<td>930</td>
<td>616</td>
<td>321</td>
<td>363</td>
<td>928</td>
<td>0</td>
</tr>
</tbody>
</table>

B. Pseudo Code

The simple idea for creating pseudo code is as follows:
1. Enter the distance matrix.
2. Determine the initial node, where the initial node will also be the last node.
3. Take the values in the distance matrix and do the calculation to find the average value. For the same value in the distance matrix, only one value is taken.
4. Use links with values below the average to find the best route.
5. Each route selected will be compared and marked to find the best value.
6. For the last route that reaches the final node, any value will be chosen so that the route forms a ring.

C. Flowchart

Flowchart for the process of the proposed algorithm can be seen in the figure below.
D. Coding Logic for Proposed Algorithm

Based on the pseudo code above, the following encoding logic will be used by the TSP simulation.

1. Inisialisasi data membaca distance matrix
2. ARRAY beban[point1,point2]
3. ARRAY pointTerpakai
4. ARRAY bebanList
5. jumlahColumn: integer
6. line_count: integer
7. point1: integer
8. point2: integer
9. costPebanding: integer
10. pointPilihan: integer
11. route:varchar
12. namaFileCsv:varchar
13. point1 < - 1
   route < - ’1’
14. nextMinCost(point1,numcol,nodeTerpakai):
   costPebanding < - 0
   FOR y in range(1, numcol):
     IF point1!=y and costPebanding == 0 and y not in nodeTerpakai:
       costPebanding < - beban[point1,y]
     END IF
     ELIF point1!=y and y not in nodeTerpakai and costPebanding > beban[point1,y]:
       costPebanding < - beban[point1,y]
     END ELIF
   NEXT y
   RETURN costPebanding
END FUNCTION
15. FUNCTION simulasi(p1<0, p2<0, jumlah_p<0, bebanTerpakai<[], route<’1’, jp<0, bebanRouting<[]):

   bebanTotal <- 0
   arrP1 <- []
   arrP1.append(p1);
   bebanTerpakai <- diff(bebanTerpakai, arrP1)
   point1 <- p1
   pointPilihan <- p1
   FOR x in range(0, jumlah_p-
   (1+len(bebanTerpakai))):
     costPebanding <- 0
     FOR y in range(1, jumlah_p):
       poin2 <- y
       IF point1!=poin2 and costPebanding == 0 and poin2 not in bebanTerpakai:
         costPebanding <- bebanRouting[point1,poin2]
         poinPilihan <- poin2
       ELIF point1!=poin2 and poin2 not in bebanTerpakai and costPebanding > bebanRouting[point1,poin2]:
         IF int(nextMinCost(poinPilihan,numcol,pointTerpakai)) > int(nextMinCost(poin2,numcol,pointTerpakai)) :
           costPebanding <- bebanRouting[point1,poin2]
           poinPilihan <- poin2
         bebanTotal<- int(bebanTotal)+int(costPebanding)
         bebanTerpakai.append(poinPilihan)
         route <- route+’ > ’+str(poinPilihan)
         poin1 <- poinPilihan
       bebanTotal<- int(bebanTotal)+int(beban[point1,1])
   NEXT x
   FOR x IN range(0, numcol-2):
     costPebanding <- 0
     pointTerpakai.append(poin1)
     PRINT(pointTerpakai)
     FOR y in range(1, numcol):
       poin2 <- y
       IF point1!=poin2 and costPebanding == 0 and poin2 not in pointTerpakai:
         costPebanding <- beban[point1,poin2]
         poinPilihan <- poin2
       NEXT y
ELIF poin1!=poin2 and poin2 not in pointTerpakai and costPebanding >= beban[poin1,poin2]:
    IF int(simulasi(poinPilihan, poin2, numcol, pointTerpakai, route, numcol, beban)) >=
    int(simulasi(poin2, poin2, int(numcol), pointTerpakai, route, numcol, beban)) :
        costPebanding <- beban[poin1,poin2] 
        poinPilihan <- poin2
    END IF
END ELIF

costTotal=int(costTotal)+int(costPebanding)
bebanList.append(costTotal)
route <- route+" > "+str(poinPilihan)
poin1 <- poinPilihan

for x in range(0, len(rs)-1):
    if x>0:
        b1 = int(rs[x-1])
        b2 = int(rs[x])
        PRINT("jarak dari"+rs[x-1]+" ke "
        tobtbn = int(beban[b1,b2])+tobtbn
        PRINT("jarak dari"+str(b2)+" ke 1 =
        tobtbn = int(beban[b1,2,1])+tobtbn
        PRINT("Total Beban = "+str(tobtbn))

rs = route.split(" > ")
totbbn = 0
for x in range(0, len(rs)-1):
    b1 = int(rs[x-1])
    b2 = int(rs[x])
    PRINT("jarak dari"+rs[x-1]+" ke "
    tobtbn = int(beban[b1,b2])+tobtbn
    PRINT("jarak dari"+str(b2)+" ke 1 =
    tobtbn = int(beban[b1,2,1])+tobtbn
    PRINT("Total Beban = "+str(tobtbn))

#selesai

Table 2. Comparison of Total Costs

<table>
<thead>
<tr>
<th>Node</th>
<th>Average Value of Distance Matrix</th>
<th>Brute Force</th>
<th>Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1038.33</td>
<td>3344</td>
<td>4736</td>
</tr>
<tr>
<td>5</td>
<td>852.1</td>
<td>3142</td>
<td>4756</td>
</tr>
<tr>
<td>6</td>
<td>908.06</td>
<td>3508</td>
<td>6376</td>
</tr>
<tr>
<td>7</td>
<td>901.09</td>
<td>4154</td>
<td>8506</td>
</tr>
<tr>
<td>8</td>
<td>852.28</td>
<td>4554</td>
<td>7753</td>
</tr>
<tr>
<td>9</td>
<td>811.47</td>
<td>4750</td>
<td>10472</td>
</tr>
<tr>
<td>10</td>
<td>780.42</td>
<td>4649</td>
<td>9590</td>
</tr>
<tr>
<td>11</td>
<td>759.74</td>
<td>4743</td>
<td>9586</td>
</tr>
<tr>
<td>12</td>
<td>743.43</td>
<td>4766</td>
<td>11139</td>
</tr>
</tbody>
</table>

Table 3. Comparison of Computation Time

<table>
<thead>
<tr>
<th>Node</th>
<th>Computation Time</th>
<th>Brute Force</th>
<th>Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.0134</td>
<td>0.0480</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.0142</td>
<td>0.0415</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.0151</td>
<td>0.0445</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.0248</td>
<td>0.0365</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.1288</td>
<td>0.0354</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.5745</td>
<td>0.0376</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>19.0063</td>
<td>0.0394</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>210.102</td>
<td>0.0391</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2383.984</td>
<td>0.0371</td>
<td></td>
</tr>
</tbody>
</table>

Whereas for computing time can be seen in table 3.

A. Comparison of Computational Time

From the table above we can make a computational time comparison chart, where a significant difference starts at node 10. At the 12th node the time needed by brute force reaches 2383.984 seconds while the proposed algorithm takes 0.0371 seconds.

B. Comparison of Total Cost

For the total cost, the results obtained through the proposed algorithm have a very big difference when compared to the results obtained through brute force. This is caused by the still not fitting the coding logic used in implementing the pseudo code that was made. So that the simulation made needs to be improved in terms of programming.

4. Simulation Results

The simulation starts with 4 nodes and finally with 12 nodes. This is because of the limitations of the hardware used in the simulation. Brute force will be a reference point in comparing the total cost and computation time obtained by this algorithm.

E. Simulation of Traveling Salesman Problem

Applications for TSP simulations are made using Python 3.7.2. This simulation is run on Microsoft Windows 10 with a computer that has an Intel i5 processor and 8 Giga bytes Random Access Memory. Data is entered in files using CSV format, by using the distance matrix created by previous research. The simulation created will show the route used, total cost and time needed to complete the computation. The solution with brute force uses source code that comes from Github (Westphahl, 2010).

4. Simulation Results

The simulation starts with 4 nodes and finally with 12 nodes. This is because of the limitations of the hardware used in the simulation. Brute force will be a reference point in comparing the total cost and computation time obtained by this algorithm.
C. Computation Time for Proposed Algorithm

In the proposed algorithm, computational time tends to decrease with increasing number of nodes. This decrease graph can be seen in Figure 4.

5. Conclusion

Based on the simulation it appears that the results obtained are still large differences compared to the brute force. There are still many weaknesses and need to be improved in terms of coding logic. On the computational time side, the proposed algorithm has a relatively stable time from node 4 to node 12. Proposed algorithms have a trend that tends to decrease with increasing number of nodes.

References


