

Tugas Kecil Penyelesaian Persoalan TSP dengan  
Algoritma *Branch and Bound*

Dery Rahman A, 13515097, K-01

Rabu, 5 April 2017

# 1 Deskripsi Permasalahan

*Travel Salesman Problem* (TSP) merupakan permasalahan dimana seorang salesman harus mengunjungi semua kota. Dan pada setiap kota tersebut salesman harus mengunjungi sekali kemudian kembali ke kota asal. Tujuannya adalah menentukan rute dengan jarak total paling minimum.

Salah satu penyelesaian TSP dapat menggunakan Algoritma *Branch and Bound*. Nilai *bound* yang dihitung dengan menggunakan *reduced cost matrix* dan Bobot Tur Lengkap. Untuk penyelesaian dengan memanfaatkan *reduced cost matrix*, digunakan matriks yang merepresentasikan graf berarah. Sedangkan untuk penyelesaian dengan Bobot Tur Lengkap, masukan berupa matriks yang merepresentasikan graf tidak berarah.

Masukan: file external dengan matriks yang merepresentasikan bobot dari graf

Keluaran:

1. Tur terpendek dan bobotnya
2. Waktu eksekusi
3. Jumlah simpul yang dibangkitkan
4. Gambar graf
5. Gambar tur terpendek (dinyatakan sebagai warna yang berbeda di dalam graf)

# 2 Kode program

```
1  #!/usr/bin/env python
2  import Queue as Q
3  import math
4
5  import time
6
7  from graph import *
8  from copy import deepcopy
9
10
11 # Convert to inf if reading INF string from file
12 def to_int(x):
13     if x == 'INF':
14         return float('inf')
15     else:
16         return int(x)
17
18
19 # Reduce matrix, and return total reducing
20 def reducing(matrix):
21     reduced = 0
22     size = len(matrix)
23     # reducing row
24     for i in range(size):
25         min_row = min(float(col) for col in matrix[i])
26         if not math.isinf(min_row):
27             reduced += min_row
28         else:
29             continue
30         for j in range(size):
31             matrix[i][j] -= min_row
32
33     # reducing col
34     for j in range(size):
35         min_col = float('inf')
36         for i in range(size):
37             if matrix[i][j] < min_col: min_col = int(matrix[i][j])
38         if not math.isinf(min_col):
39             reduced += min_col
40         else:
41             continue
```

```

42         for i in range(size):
43             matrix[i][j] -= min_col
44
45     return reduced
46
47
48 # Print matrix for debugging
49 # def print_matrix(matrix):
50 #     for i in range(len(matrix)):
51 #         for j in range(len(matrix[i])):
52 #             print '%3s' % matrix[i][j],
53 #         print
54
55
56 # Update matrix from f to t
57 def bound(f, t, matrix):
58     a = matrix[f][t]
59     matrix[t][f] = float('inf')
60     size = len(matrix)
61     matrix[f] = [float('inf') for col in matrix[f]]
62     for j in range(size):
63         matrix[j][t] = float('inf')
64     r = reducing(matrix)
65     return a + r
66
67
68 # Return list of unvisited node
69 def node_child(matrix):
70     l = []
71     for i in range(len(matrix)):
72         for j in range(len(matrix[i])):
73             if not math.isinf(matrix[i][j]):
74                 l += [i]
75     l = list(set(l))
76     return l
77
78
79 # Solve TSP recursively
80 def solve(q, state_num):
81     cur_state = q.get()
82     reducing_state = cur_state[0]
83     node_state = cur_state[1]
84     matrix_state = cur_state[2]
85     node_target = node_child(matrix_state)
86     if len(node_target) == 1:
87         node_state.append(node_state[0])
88         node_state = [elem + 1 for elem in node_state]
89         return reducing_state, node_state, state_num
90     else:
91         state_num += len(node_target) - 1
92         for i in node_target:
93             if i == node_state[-1]: continue
94             matrix_temp = deepcopy(matrix_state)
95             total = reducing_state + bound(node_state[-1], i, matrix_temp)
96             q.put((total, node_state + [i], matrix_temp))
97         return solve(q, state_num)
98
99
100 # Read user input
101 filename = raw_input("Masukan nama file : ")
102 node_init = int(raw_input("Masukan node awal : "))
103
104 # Read file and make a matrix
105 matrix = open("../data/" + filename, "r").read()
106 matrix = [item.split() for item in matrix.split('\n')[:-1]]
107 matrix = [[to_int(column) for column in row] for row in matrix]
108
109 # create initial graph
110 create_graph(matrix, [], True)
111 matrix_final = deepcopy(matrix)
112
113 # Node 1..N is 0..N-1
114 node_init -= 1
115
116 # Add reducing state to priority queue
117 q = Q.PriorityQueue()

```

```

118 c0 = reducing(matrix)
119 state_num = 1
120 q.put((c0, [node_init], matrix))
121
122 start_time = time.time()
123 # Solve TSP :)
124 result = solve(q, state_num)
125 print("Total bobot tereduksi : %.2f" % result[0])
126 print("Jalur yang ditempuh : ",
127       print result[1])
128 print("Simpul yang dibangkitkan : %d" % result[2])
129 print("Waktu yang diperlukan : %.6s detik" % (time.time() - start_time))
130
131 # create graph after TSP algorithm
132 create_graph(matrix_final, result[1], True)

```

Listing 1: Implementasi *Branch and Bound* dengan *Reduction Cost Matrix*

```

1 #!/usr/bin/env python
2 import Queue as Q
3 import math
4
5 import time
6
7 from graph import *
8 from copy import deepcopy
9
10
11 # Convert to inf if reading INF string from file
12 def to_int(x):
13     if x == 'INF':
14         return float('inf')
15     else:
16         return int(x)
17
18 # Lower bound
19 def lower_bound(matrix):
20     table_two_minimum = []
21     sum = 0
22     for row in matrix:
23         m1, m2 = float('inf'), float('inf')
24         for x in row:
25             if x <= m1:
26                 m1, m2 = x, m1
27             elif x < m2:
28                 m2 = x
29         sum+=m1+m2
30         table_two_minimum.append([m1,m2])
31     return sum, table_two_minimum
32
33 # Calculate cost
34 def cost(node, matrix):
35     sum = 0
36     remain = list(set(range(len(matrix)))-set(node))
37     for i in range(1,len(node)-1):
38         sum+=matrix[node[i]][node[i+1]]+matrix[node[i]][node[i-1]]
39     if not node[0]==node[-1]:
40         sum+=matrix[node[0]][node[1]]+matrix[node[-1]][node[-2]]
41     m1=0
42     m2=0
43     if remain :
44         m1 = float('inf')
45         m2 = float('inf')
46         for i in remain:
47             if matrix[node[0]][i]<m1:
48                 m1 = matrix[node[0]][i]
49             if matrix[node[-1]][i]<m2:
50                 m2 = matrix[node[-1]][i]
51         sum+=m1+m2
52
53     if sum == 0: # if still in first node
54         remain+=node[0]
55     for i in remain:
56         m1, m2 = float('inf'), float('inf')
57         for j in range(len(matrix)):

```

```

58         if matrix[i][j] <= m1:
59             m1, m2 = matrix[i][j], m1
60         elif matrix[i][j] < m2:
61             m2 = matrix[i][j]
62         sum+=m1+m2
63
64         return float(sum)/2
65
66     def node_child(node_state, matrix):
67         return list(set(range(len(matrix))))-set(node_state))
68
69
70 # Solve TSP using priority queue
71 def solve(q, matrix, state_num):
72     cost_res = 0
73     node_state_res = []
74     B = float('inf')
75     while(not q.empty()):
76         cur_state = q.get()
77         curr_cost = cur_state[0]
78         node_state = cur_state[1]
79         node_target = node_child(node_state, matrix)
80
81         if not node_target:
82             if not node_state[0]==node_state[-1]:
83                 node_next = node_state + [node_state[0]]
84                 cost_next = curr_cost+matrix[node_state[-1]][node_state[0]]
85                 q.put((cost_next, node_next))
86             else :
87                 if curr_cost<B :
88                     node_state_res = node_state
89                     cost_res = curr_cost
90                     B = curr_cost
91                     store = []
92                     while not q.empty():
93                         temp = q.get()
94                         if abs(temp[0] - B)<=0.1:
95                             store.append(temp)
96                     for elem in store:
97                         q.put(elem)
98
99                 state_num += len(node_target)
100                 for i in node_target:
101                     node_next = node_state + [i]
102                     cost_next = cost(node_next, matrix)
103                     if cost_next<=B:
104                         q.put((cost_next, node_next))
105
106         node_state_res = [elem + 1 for elem in node_state_res]
107         return cost_res, node_state_res, state_num
108
109
110 # Read user input
111 filename = raw_input("Masukan nama file : ")
112 node_init = int(raw_input("Masukan node awal : "))
113
114 # Read file and make a matrix
115 matrix = open("../data/" + filename, "r").read()
116 matrix = [item.split() for item in matrix.split('\n')[:-1]]
117 matrix = [[to_int(column) for column in row] for row in matrix]
118
119 # create initial graph
120 create_graph(matrix)
121 matrix_final = deepcopy(matrix)
122
123 # Node 1..N is 0..N-1
124 node_init -= 1
125
126 # Add reducing state to priority queue
127 q = Q.PriorityQueue()
128 low_bound = cost([node_init], matrix)
129 state_num = 1
130 q.put((low_bound, [node_init]))
131
132 start_time = time.time()
133 # Solve TSP :)

```

```

134 result = solve(q,matrix,state_num)
135 print("Bobot tur lengkap : %.2f" % result[0])
136 print "Jalur yang ditempuh : ",
137 print result[1]
138 print("Simpul yang dibangkitkan : %d" % result[2])
139 print("Waktu yang diperlukan : %.6s detik" % (time.time() - start_time))
140
141 # create graph after TSP algorithm
142 create_graph(matrix_final,result[1])

```

Listing 2: Implementasi *Branch and Bound* dengan Bobot Tur Lengkap

```

1 import math
2 import networkx as nx
3 import matplotlib.pyplot as plt
4
5
6 def create_graph(matrix, result=None, digraph=False):
7
8     if result is None:
9         result = []
10
11     if digraph:
12         G = nx.MultiDiGraph()
13     else :
14         G = nx.MultiGraph()
15     G.add_nodes_from(range(1,len(matrix)+1))
16     for i in range(len(matrix)):
17         for j in range(len(matrix)):
18             if not math.isinf(float(matrix[i][j])) :
19                 G.add_edge(i+1,j+1,length=matrix[i][j])
20
21     pos = nx.shell_layout(G)
22     edge_labels=dict([(u,v,d['length']) for u,v,d in G.edges(data=True)])
23
24     result = [(result[i],result[i+1]) for i in range(len(result)-1)]
25     nx.draw_networkx_labels(G,pos, font_color='white', font_size=5)
26     nx.draw_networkx_nodes(G,pos,node_size=80, node_color = 'black')
27     nx.draw_networkx_edges(G,pos,width=0.3, edge_color="red")
28     nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, label_pos=0.9,
29                                font_size=4)
30     if(result):
31         nx.draw_networkx_edges(G, pos, result, width=1, edge_color="green")
32         plt.axis('off')
33     if(not result):
34         plt.savefig("../output/initial_graph.svg")
35     else:
36         plt.savefig("../output/final_graph.svg")

```

Listing 3: Implementasi fungsi untuk menggambar *Graph*

### 3 Screenshoot Output

```

+ × src: python
dery@dery-X450JF:~/PycharmProjects/TSP/src$ python reduction_cost.py
Masukan nama file : tc1.in
Masukan node awal : 1
Total bobot tereduksi : 171.00
Jalur yang ditempuh : [1, 8, 3, 2, 6, 4, 7, 5, 1]
Simpul yang dibangkitkan : 29
Waktu yang diperlukan : 0.0031 detik
dery@dery-X450JF:~/PycharmProjects/TSP/src$

```

Gambar 1: Hasil *test case 1* dari RCM

```
+ x src: python
dery@dery-X450JF:~/PycharmProjects/TSP/src$ python reduction_cost.py
Masukan nama file : tc2.in
Masukan node awal : 1
Total bobot tereduksi : 451.00
Jalur yang ditempuh : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1]
Simpul yang dibangkitkan : 46
Waktu yang diperlukan : 0.0087 detik
dery@dery-X450JF:~/PycharmProjects/TSP/src$
```

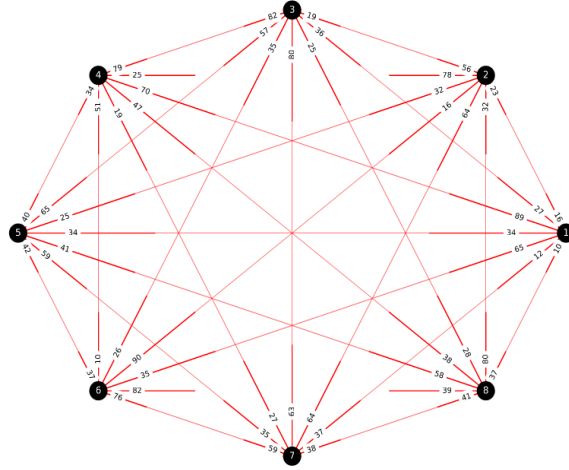
Gambar 2: Hasil *test case 2* dari RCM

```
+ x src: python
dery@dery-X450JF:~/PycharmProjects/TSP/src$ python complete_tour.py
Masukan nama file : tc3.in
Masukan node awal : 1
Bobot tur lengkap : 183.00
Jalur yang ditempuh : [1, 7, 4, 5, 2, 6, 3, 8, 1]
Simpul yang dibangkitkan : 115
Waktu yang diperlukan : 0.0018 detik
dery@dery-X450JF:~/PycharmProjects/TSP/src$
```

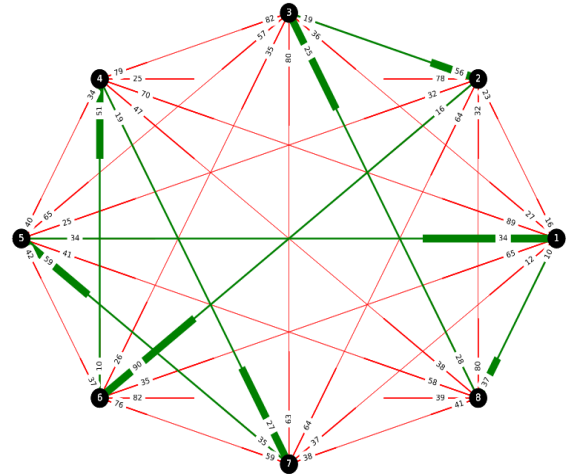
Gambar 3: Hasil *test case 1* dari Bobot Tur Lengkap

```
+ x src: python
dery@dery-X450JF:~/PycharmProjects/TSP/src$ python complete_tour.py
Masukan nama file : tc4.in
Masukan node awal : 1
Bobot tur lengkap : 250.00
Jalur yang ditempuh : [1, 6, 2, 7, 3, 8, 4, 9, 5, 10, 1]
Simpul yang dibangkitkan : 371520
Waktu yang diperlukan : 53.853 detik
dery@dery-X450JF:~/PycharmProjects/TSP/src$
```

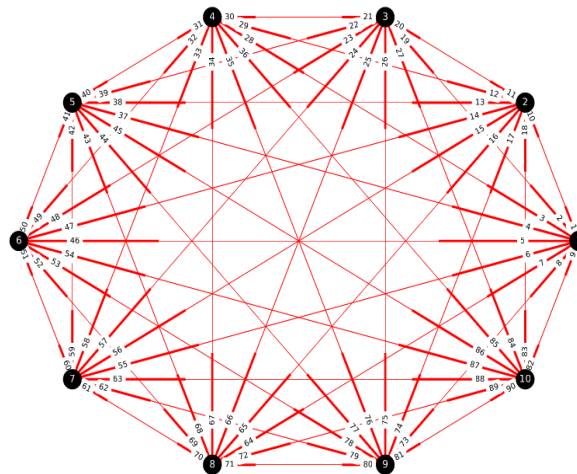
Gambar 4: Hasil *test case 2* dari Bobot Tur Lengkap



Gambar 5: Graf *test case 1*

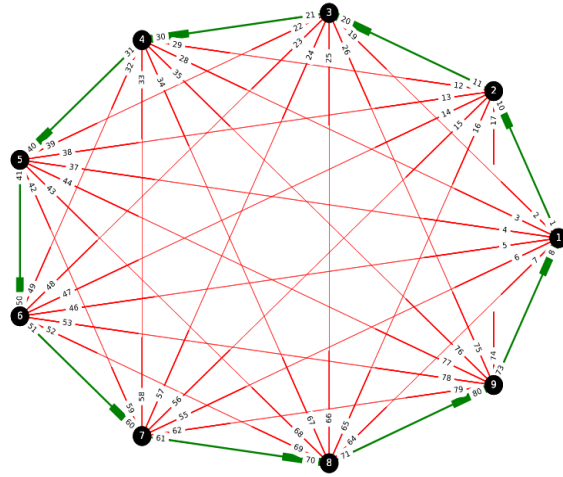


Gambar 6: Graf hasil *test case 1* dari RCM

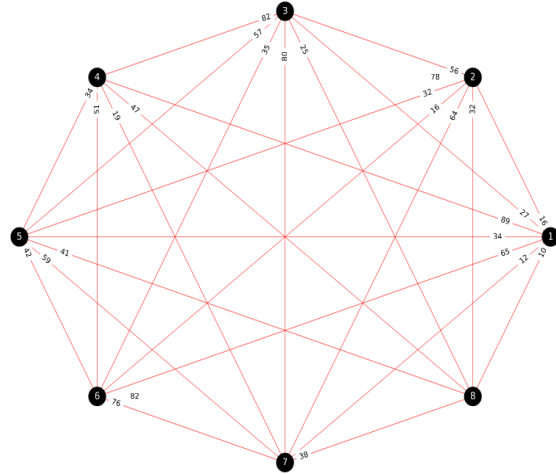


Gambar 7: Graf *test case 2*

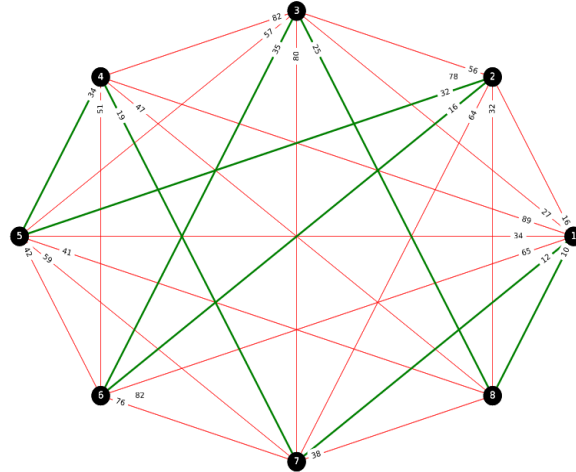




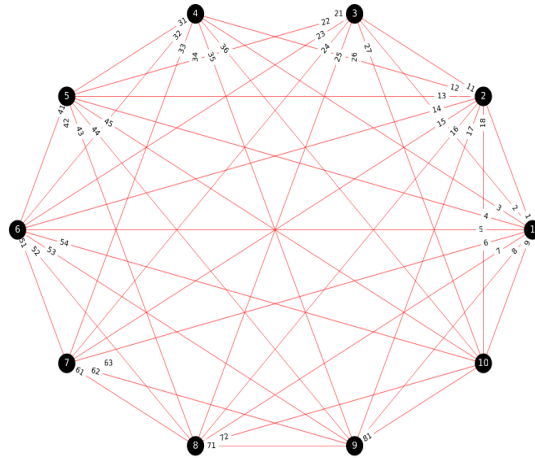
Gambar 8: Graf hasil *test case 2* dari RCM



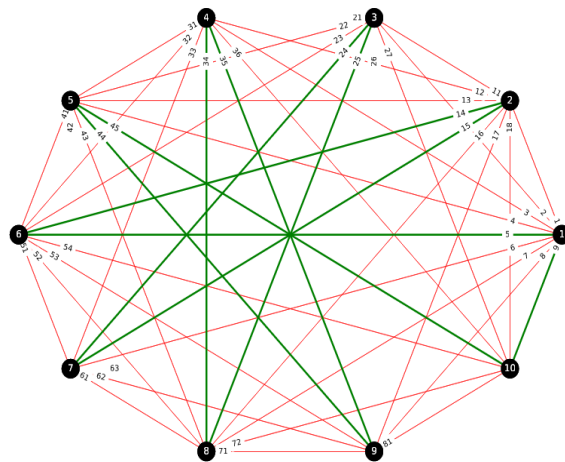
Gambar 9: Graf *test case 1*



Gambar 10: Graf hasil *test case 1* dari Bobot Tur Lengkap



Gambar 11: Graf *test case 2*



Gambar 12: Graf hasil *test case 2* dari Bobot Tur Lengkap