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 degan 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

```
#!/usr/bin/env python
import Queue as Q
import math

import time

from graph import *
from copy import deepcopy


# Convert to inf if reading INF string from file
def to_int(x):
    if x == 'INF':
        return float('inf')
    else:
        return int(x)


# Reduce matrix, and return total reducing
def reducing(matrix):
    reduced = 0
    size = len(matrix)
    # reducing row
    for i in range(size):
        min_row = min(float(col) for col in matrix[i])
        if not math.isinf(min_row):
            reduced += min_row
        else:
            continue
        for j in range(size):
            matrix[i][j] -= min_row

    # reducing col
    for j in range(size):
        min_col = float('inf')
        for i in range(size):
            if matrix[i][j] < min_col: min_col = int(matrix[i][j])
        if not math.isinf(min_col):
            reduced += min_col
        else:
            continue
```

```
        for i in range(size):
            matrix[i][j] -= min_col

    return reduced


# Print matrix for debugging
# def print_matrix(matrix):
#     for i in range(len(matrix)):
#         for j in range(len(matrix[i])):
#             print '%3s' % matrix[i][j],
#         print


# Update matrix from f to t
def bound(f, t, matrix):
    a = matrix[f][t]
    matrix[t][f] = float('inf')
    size = len(matrix)
    matrix[f] = [float('inf') for col in matrix[f]]
    for j in range(size):
        matrix[j][t] = float('inf')
    r = reducing(matrix)
    return a + r


# Return list of unvisited node
def node_child(matrix):
    l = []
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if not math.isinf(matrix[i][j]):
                l += [i]
    l = list(set(l))
    return l


# Solve TSP recursively
def solve(q,state_num):
    cur_state = q.get()
    reducing_state = cur_state[0]
    node_state = cur_state[1]
    matrix_state = cur_state[2]
    node_target = node_child(matrix_state)
    if len(node_target) == 1:
        node_state.append(node_state[0])
        node_state = [elem + 1 for elem in node_state]
        return reducing_state, node_state, state_num
    else:
        state_num+=len(node_target)-1
        for i in node_target:
            if i == node_state[-1]: continue
            matrix_temp = deepcopy(matrix_state)
            total = reducing_state + bound(node_state[-1], i, matrix_temp)
            q.put((total, node_state + [i], matrix_temp))
        return solve(q,state_num)


# Read user input
filename = raw_input("Masukan nama file : ")
node_init = int(raw_input("Masukan node awal : "))

# Read file and make a matrix
matrix = open("../data/" + filename, "r").read()
matrix = [item.split() for item in matrix.split('\n')[:-1]]
matrix = [[to_int(column) for column in row] for row in matrix]

# create initial graph
create_graph(matrix,[],True)
matrix_final = deepcopy(matrix)

# Node 1..N is 0..N-1
node_init -= 1

# Add reducing state to priority queue
q = Q.PriorityQueue()
```

```
c0 = reducing(matrix)
state_num = 1
q.put((c0, [node_init], matrix))

start_time = time.time()
# Solve TSP :)
result = solve(q,state_num)
print("Total bobot tereduksi : %.2f" % result[0])
print "Jalur yang ditempuh : ",
print result[1]
print("Simpul yang dibangkitkan : %d" % result[2])
print("Waktu yang diperlukan : %.6s detik" % (time.time() - start_time))

# create graph after TSP algorithm
create_graph(matrix_final,result[1],True)
```

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

```
#!/usr/bin/env python
import Queue as Q
import math

import time

from graph import *
from copy import deepcopy


# Convert to inf if reading INF string from file
def to_int(x):
    if x == 'INF':
        return float('inf')
    else:
        return int(x)

# Lower bound
def lower_bound(matrix):
    table_two_minimum = []
    sum = 0
    for row in matrix:
        m1, m2 = float('inf'), float('inf')
        for x in row:
            if x <= m1:
                m1, m2 = x, m1
            elif x < m2:
                m2 = x
        sum+=m1+m2
        table_two_minimum.append([m1,m2])
    return sum, table_two_minimum

# Calculate cost
def cost(node, matrix):
    sum = 0
    remain = list(set(range(len(matrix)))-set(node))
    for i in range(1,len(node)-1):
        sum+=matrix[node[i]][node[i+1]]+matrix[node[i]][node[i-1]]
    if not node[0]==node[-1]:
        sum+=matrix[node[0]][node[1]]+matrix[node[-1]][node[-2]]
        m1=0
        m2=0
        if remain :
            m1 = float('inf')
            m2 = float('inf')
            for i in remain:
                if matrix[node[0]][i]<m1:
                    m1 = matrix[node[0]][i]
                if matrix[node[-1]][i]<m2:
                    m2 = matrix[node[-1]][i]
        sum+=m1+m2

    if sum == 0: # if still in first node
        remain+=[node[0]]
    for i in remain:
        m1, m2 = float('inf'), float('inf')
        for j in range(len(matrix)):
```

```
            if matrix[i][j] <= m1:
                m1, m2 = matrix[i][j], m1
            elif matrix[i][j] < m2:
                m2 = matrix[i][j]
        sum+=m1+m2

    return float(sum)/2

def node_child(node_state,matrix):
    return list(set(range(len(matrix)))-set(node_state))


# Solve TSP using priority queue
def solve(q,matrix,state_num):
    cost_res = 0
    node_state_res = []
    B = float('inf')
    while(not q.empty()):
        cur_state = q.get()
        curr_cost = cur_state[0]
        node_state = cur_state[1]
        node_target = node_child(node_state,matrix)

        if not node_target:
            if not node_state[0]==node_state[-1]:
                node_next = node_state + [node_state[0]]
                cost_next = curr_cost+matrix[node_state[-1]][node_state[0]]
                q.put((cost_next,node_next))
            else :
                if curr_cost<B :
                    node_state_res = node_state
                    cost_res = curr_cost
                B = curr_cost
                store = []
                while not q.empty():
                    temp = q.get()
                    if abs(temp[0] - B)<=0.1:
                        store.append(temp)
                for elem in store:
                    q.put(elem)

        state_num += len(node_target)
        for i in node_target:
            node_next = node_state + [i]
            cost_next = cost(node_next,matrix)
            if cost_next<=B:
                q.put((cost_next, node_next))

    node_state_res = [elem + 1 for elem in node_state_res]
    return cost_res, node_state_res, state_num


# Read user input
filename = raw_input("Masukan nama file : ")
node_init = int(raw_input("Masukan node awal : "))

# Read file and make a matrix
matrix = open("../data/" + filename, "r").read()
matrix = [item.split() for item in matrix.split('\n')[:-1]]
matrix = [[to_int(column) for column in row] for row in matrix]

# create initial graph
create_graph(matrix)
matrix_final = deepcopy(matrix)

# Node 1..N is 0..N-1
node_init -= 1

# Add reducing state to priority queue
q = Q.PriorityQueue()
low_bound = cost([node_init],matrix)
state_num = 1
q.put((low_bound, [node_init]))

start_time = time.time()
# Solve TSP :)
```

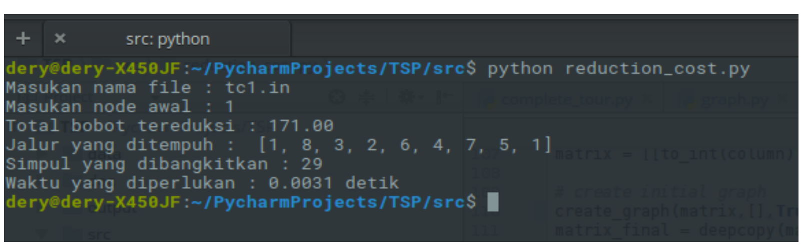
```python
result = solve(q,matrix,state_num)
print("Bobot tur lengkap : %.2f" % result[0])
print "Jalur yang ditempuh : ",
print result[1]
print("Simpul yang dibangkitkan : %d" % result[2])
print("Waktu yang diperlukan : %.6s detik" % (time.time() - start_time))

# create graph after TSP algorithm
create_graph(matrix_final,result[1])
```

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

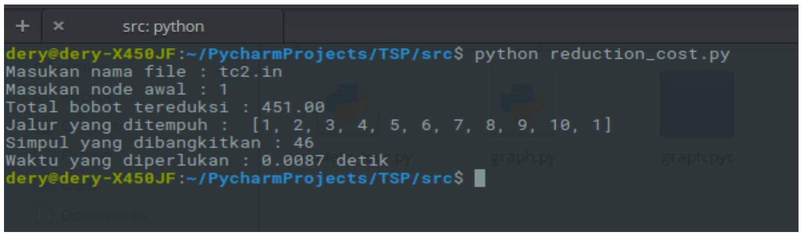
```python
import math
import networkx as nx
import matplotlib.pyplot as plt


def create_graph(matrix, result=None, digraph=False):

    if result is None:
        result = []
    if digraph:
        G = nx.MultiDiGraph()
    else :
        G = nx.MultiGraph()
    G.add_nodes_from(range(1,len(matrix)+1))
    for i in range(len(matrix)):
        for j in range(len(matrix)):
            if not math.isinf(float(matrix[i][j])) :
                G.add_edge(i+1,j+1,length=matrix[i][j])

    pos = nx.shell_layout(G)
    edge_labels=dict([((u,v,),d['length']) for u,v,d in G.edges(data=True)])

    result = [(result[i],result[i+1]) for i in range(len(result)-1)]
    nx.draw_networkx_labels(G,pos, font_color='white', font_size=5)
    nx.draw_networkx_nodes(G,pos,node_size=80, node_color = 'black')
    nx.draw_networkx_edges(G,pos,width=0.3, edge_color="red")
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, label_pos=0.9, font_size=4)
    if(result):
        nx.draw_networkx_edges(G, pos, result, width=1, edge_color="green")
    plt.axis('off')
    if(not result):
        plt.savefig("../output/initial_graph.svg")
    else:
        plt.savefig("../output/final_graph.svg")
```

Listing 3: Implementasi fungsi untuk menggambar *Graph*

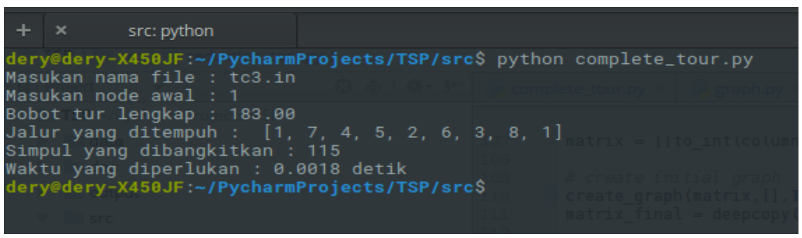
3 *Screenshoot Output*



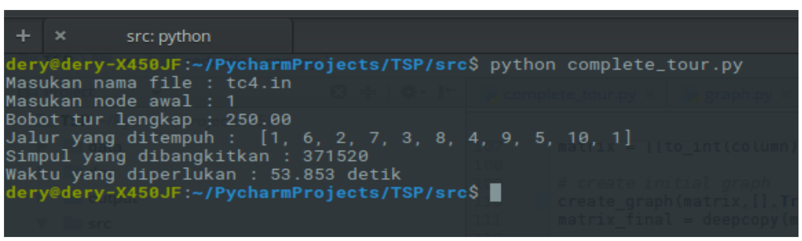
Gambar 1: Hasil *test case 1* dari RCM



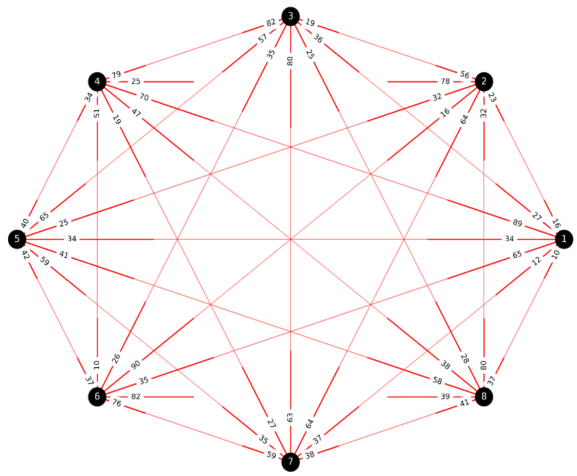
Gambar 2: Hasil *test case 2* dari RCM



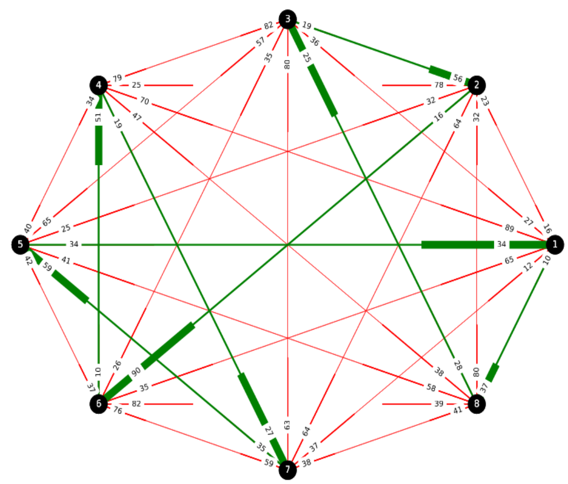
Gambar 3: Hasil *test case 1* dari Bobot Tur Lengkap



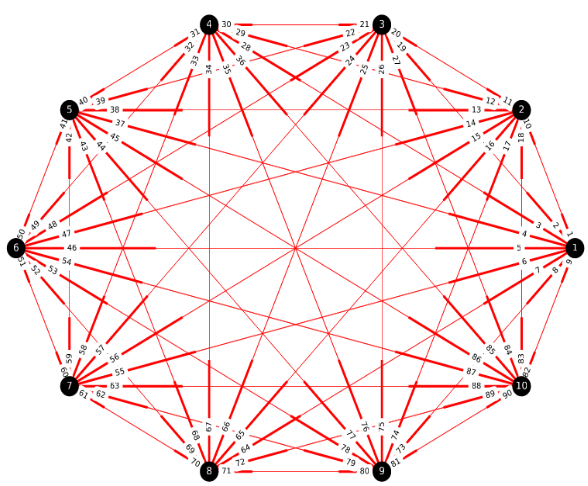
Gambar 4: Hasil *test case 2* dari Bobot Tur Lengkap



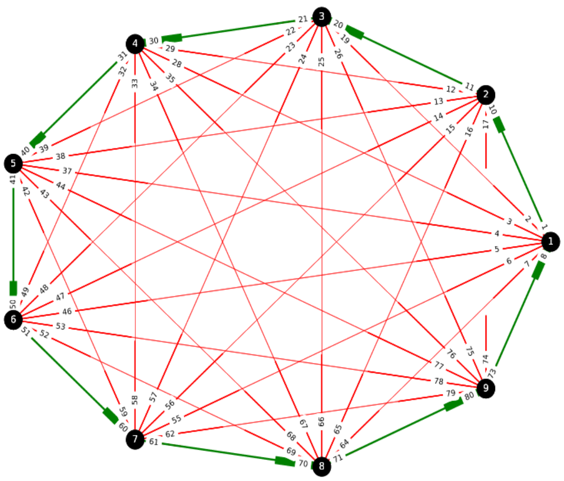
Gambar 5: Graf *test case 1*



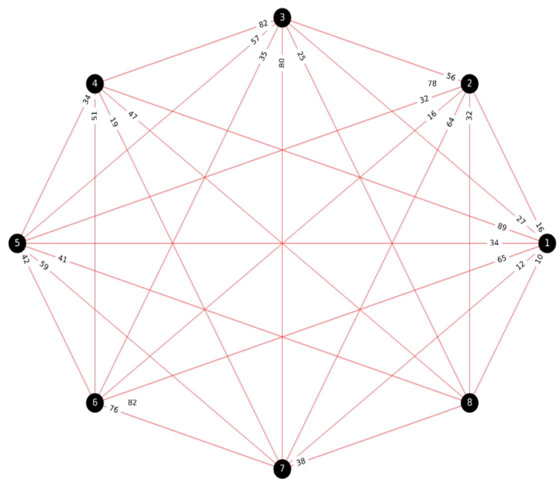
Gambar 6: Graf hasil *test case 1* dari RCM



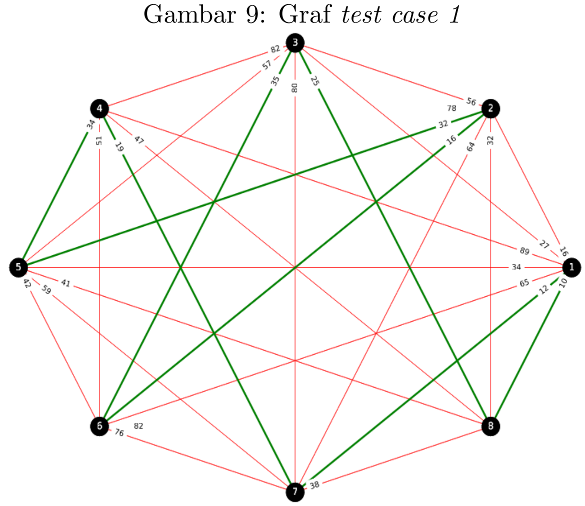
Gambar 7: Graf *test case 2*



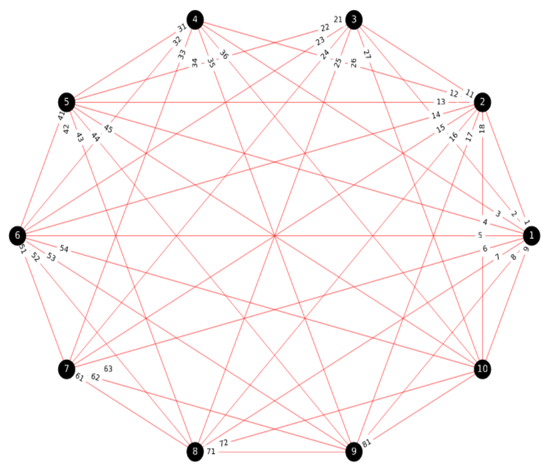
Gambar 8: Graf hasil *test case 2* dari RCM



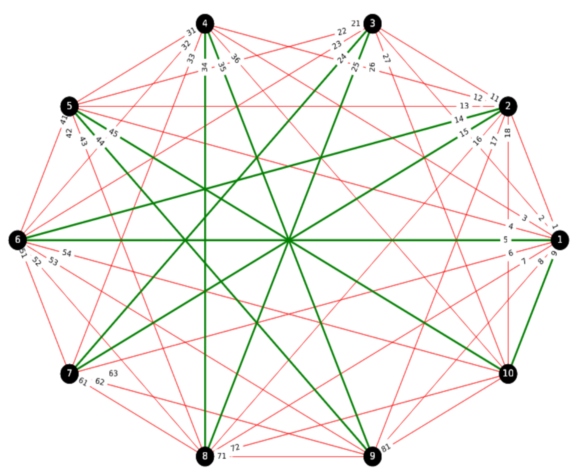
Gambar 9: Graf *test case 1*



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