Penerapan Algoritma *Branch and Bound* dalam Pemacahan *Travelling Salesman Problem* (TSP) dalam Graf Lengkap

Irfan Ariq
Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13515112@std.stei.itb.ac.id

***Abstrak*—*Travelling Salesman Problem* atau yang biasa disingkat TSP merupakan salah satu persoalan dalam graf. Dalam permasalahan TSP ini terdapat seorang *salesman* yang memiliki kewajiban untuk mengunjungi semua kota dengan batasan semua kota hanya boleh dikunjungi tepat sekali dan *salesman* tersebut harus kembali ke kota awal dengan jarak tempuh seminimum mungkin.**

Algoritma *Branch and Bound* merupakan salah satu algoritma yang dapat menyelesaikan permasalahan TSP. terdapat dua cara dalam penentuan *cost*-nya. Cara pertama adalah *reduced cost matrix* dan cara kedua adalah bobot tur lengkap.

Keywords—Travelling Salesman Problem, Algoritma Branch and Bound, Reduced Cost Matrix, Bobot Tur Lengkap.

I. PENDAHULUAN

Menurut Suprojo dan Purwadi, 1982 dalam Tarmizi, 2005, bahwa secara matematis optimasi adalah cara mendapatkan nilai ekstrim bik minimum ataupun makasimum dari suatu fungsi tertentu dengan faktor – faktor pembatasnya. Dalam permasalahan TSP, kita perlu mencari rute dengan batasan – batasan tertentu dan jarak tempuh paling kecil. Oleh karena itu TSP merupakan salah satu persoalan optimasi.

Seorang *salesman* memiliki kewajiban untuk menawarkan barang dari suatu tempat ke tempat lain. Untuk pindah dari suatu tempat ke tempat lain, *salesman* harus memikirkan waktu yang dibutuhkan, jarak yang akan ditempuh, serta biaya yang diperlukan unuk berpindah tempat. Efisien atau tidaknya perpindahan tempat dipengaruhi oleh tiga hal tersebut. Perpindahan tempat dapat dikatakan efisien apabila waktu yang dibutuhkan singkat, jarak yang ditempuh tidak jauh, dan biaya yang dikeluarkan sedikit. Efisiensi yang baik dari pencarian rute ini merupakan permasalahan optimasi, dimana seorang *salesman* harus mencari rute paling efisien dengan waktu tempuh paling singkat, jarak tempuh paling pendek, dan biaya yang dibutuhkan paling sedikit. Namun terkadang kita dapat menekankan pada batasan jarak tempuh saja sehingga waktu tempuh dan biaya yang diperlukan dapat daibaikan atau dianggap berbanding lurus dengan jarak tempuh.

Lalu, bagaimana mencari rute perpindahan tempat dengan total jarak tempuh yang harus dilalui terpendek? Kita dapat melihat bahwa permasalahan ini memiliki model yang sama dengan TSP sehingga dapat dimodelkan menjadi permaslahan TSP. Dengan memodelkan menjadi TSP kita hanya perlu mencari algoritma yang dapat menyelesaikan permaslahan TSP dengan efisien. Disini kita akan menggunakan algoritma *branch and bound* untuk menyelesaikan permasalahan tersebut.

II. TEORI DASAR

A. *Travelling Salesman Problem (TSP)*

Travelling Salesman Problem atau yang biasa disebut TSP pertama kali diperkenalkan pada tahun 1948 oleh Rand. Persoalan TSP ini melibatkan seorang *salesman* yang memiliki kewajiban menawarkan produk dagangannya ke sejumlah kota sehingga *salesman* tersebut harus melakukan kunjungan ke sejumlah kota. Namun terdapat beberapa ketentuan yang diberikan kepada *salesman* tersebut dalam melakukan perjalanannya. Beberapa ketentuan yang diberikan seperti *salesman* hanya boleh mengunjungi setiap kota tepat satu kali dan ia harus kembail ke kota dimana ia mengawali perjalanannya. Ketentutan lainnya ialah rute perjalanan yang ditempuh haruslah memiliki jarak tempuh total paling minimum.

Jika dilihat dari sejarahnya, permasalahn TSP ini dapat ditelusuri dari tahun 1759 pada saat Euler mempelajari *Knight Tour's Problem*, tahun 1856 saat Kirkman mempelajari persoalan grafik polihedron dan Hamilton yang membuat game Icosian yang bertujuan mencari jalur sirkuit berbasis grafik polihedron dengan syarat tertentu. Namun istilah TSP sendiri diperkirakan berasal pada tahun 1930an dari buku yang diterbitkan oleh seorang veteran salesman di Jerman.

TSP merupakan salah satu persoalan optimasi dimana kita harus mencari rute dengan total jarak tempuh paling minimum. Selain termasuk dalam kategori persoalan optimasi, TSP juga merupakan persoalan kombinatorial dimana dapat menghasilkan lebih dari satu solusi.

B. *Algoritma Branch and Bound*

Branch and Bound (B&B) merupakan salah satu metode pencarian di dalam ruang solusi secara sistematik. Ruang solusi ini digambarkan dalam bentuk pohon ruang status. Pembentukan pohon ruang status pada algoritma ini mirip dengan pembentukan pohon ruang status pada algoritma *Breadth First Search* (BFS). Setiap simpul pada pohon ruang status akan diberikan sebuah nilai yang disebut cost yang berguna untuk mempercepat pencarian ke simpul solusi. Simpul yang akan diekspansi pun tidak berdasarkan urutan kebangkitan seperti BFS pada umumnya, melainkan berdasarkan *cost* yang ada pada setiap simpul. Masalah simpul mana yang akan diekspansi terlebih dahulu dapat berbeda – beda tergantug persoalan apa yang akan diselesaikan. Nilai dari cost tiap simpul merupakan taksiran *cost* termurah dari simpul ke-i sampai simpul solusi.

Tidak seperti algoritma BFS yang hanya perlu sebuah queue, algoritma branch and bound membutuhkan priority queue untuk menyimpan simpul – simpul mana saja yang harus diekspansi. Simpul – simpul tersebut akan diurutkan berdasarkan *cost* simpul tersebut.

Dalam algoritam B&B ini, apabila kita sudah mendapat suatu solusi, kita hanya perlu mengekspansi simpul yang masih layak untuk diekspansi. Syarat sebuah simpul layak diekspansi adalah nilai *cost*-nya tidak lebih dari *cost* pada simpul solusi.

Dalam menyelesaikan persoalan TSP, terdapat dua cara dalam menentukan *cost*-nya. Cara yang pertama disebut *reduced cost matrix* sedangkan cara yang kedua disebut bobot tur lengkap.

- Cara menghitung *cost* menggunakan *reduced cost matrix*

Untuk mengitung *cost* menggunakan cara *reduced cost matrix* tentu kita harus membuat matriks ketetanggaannya terlebih dahulu. Setelah itu kita menentukan dari kota atau simpul mana kita akan berawal. Kota atau simpul asal ini akan menjadi akar pada pohon ruang status.

Matriks dikatakan tereduksi apabila setiap baris dan kolom paling tidak memiliki satu nilai 0. Pada simpul akar pohon ruang status matriks akan direduksi dari matriks ketetanggan pada graf. Untuk mendapatkan paling tidak satu elemen dari baris dan kolom bernilai 0, kita harus mengurangi setiap elemen baris dan kolom dengan elemen terkecil dari baris dan kolom itu sendiri. Jumlah total elemen pengurang dari semua baris dan kolom menjadi batas bawah atau *cost* pada simpul akar.

Selanjutnya, misalkan M merupakan sebuah matriks tereduksi untuk simpul A dan B merupakan anak dari simpul A dari pohon ruang status sedemikian sehingga sisi (A, B) pada pohon ruang status berkoresponden dengan sisi (i, j) pada perjalanan. Jika M bukan merupakan simpul daun maka *reduced cost matrix* dari simpul M dapat dihitung dengan carai berikut ini.

1. Ubah semua nilai pada baris i dan kolom j menjadi tak hingga.
2. Ubah M(j,i) menjadi tak hingga. Ini berutujan untuk mecegah adanya sirkular namun belum semua kota dikujungi.
3. Reduksi kembail matriks M sehingga setiap kolom dan baris memiliki setidaknya satu nilai 0. Namun baris dan kolom yang direduksi selain kolom atau baris dengan nilai tak hinnga semua.

Berikut ini merupakan rumus yang digunakan untuk mennetukan *cost*.

$$c(B) = c(A) + M(i,j) + r,$$

yang dalam hal ini :

c(B) = cost minimum yang dilalui simpul B.

c(A) = cost dari simpul yang melalui A, yang dalam hal ini A adalah simpul orang tua dari B/

M(i,j) = jarak siis (i,j) pada graf yang berkorespondensi dengan sisi (B, A) pada pohon ruang status.

r = jumlah semua pengurang pada proposal.

- Cara menghitung *cost* menggunakan bobot tur lengkap

Berbeda dengan cara *reduced cost matrix* yang harus membuat matriks ketetanggaannya, dengan bobot tur lengkap kita tidak harus membuat matriks ketetaanggaannya. Berikut ini merupakan rumus untuk menentukan cost menggunakan cara bobot tur lengkap.

$$M \equiv cost = bobot\ minimum\ tur\ lengkap \geq \frac{1}{2}\sum bobot\ sisi\ i1 + bobot\ sisi\ i2$$

Bobot i1 dan bobot i2 merupakan sisi yang bersisian dengan simpul i dengan bobot paling minimum. Tentu sisi i1 dan sisi i2 merupakan sisi yang berbeda. kita pun harus menjumlahkan semua bobot i1 dan bobot i2 pada semua simpul yang ada pada graf. M dapat digunakan sebagai fungsi *bounding* untuk menghitung cost pada setiap simpul yang ada pada pohon ruang status.

Pada saat percabangan simpul, perhitunga *cost* sedikit berbeda. Bobot i1 dan bobot i2 bisa saja digantikan dengan bobot dari kota asal ke kota tujuan. Contohnya pada saat menghitung suatu simpul yang berjalan dari kota A ke kota D. Sehingga pada saat mencari bobot i1 pada A dan D akan diganti menjadi bobot sisi yang bersisian dengan simpul A dan D.

III. Implementasi Algoritma *Branch and Bound* dalam Pemecahan Persoalan TSP

A. *Pemecahan TSP dengan Bobot Tur Lengkap*

Sebuah permasalahan TSP yang mengharuskan kita untuk mecari rute dengan total jarak tempuh terpendek dengan tambahan syarat yaitu harus kembali lagi ke kota awal tempat kita memulai perjalanan. Berikut deskripsi persoalan TSP yang lebih rinci.

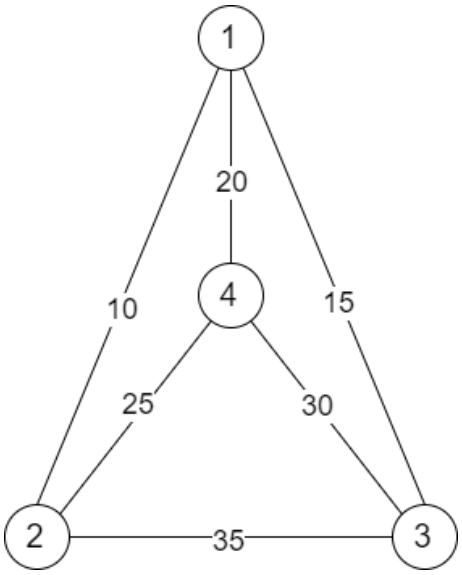
Seorang *salesman* yang diberikan tugas untuk pergi ke sejumlah kota untuk menawarkan produk dagangannya. Terdapat 4 kota yang wajib dikunjungi oleh *salesman* tersebut, diantaranya kota ke-1, kota ke-2, kota ke-3, dan kota ke-4. *Salesman* tersebut akan memulai perjalanannnya dari kota ke-1.

Jarak tempuh dari jalan antara satu kota dengan kota yang lain berbeda – beda. Jarak tempuh kota asal ke kota tujuan dan kota tujuan ke kota asal bernilai sama. Contohnya jarak tempuh dari kota A ke kota B sama dengan jarak tempuh dari kota B ke kota A. Berikut ini merupakan rician jarak antara satu kota dengan kota yang lainnnya.

- Jarak tempuh dari jalan kota ke-1 ke kota ke-2 adalah 10.
- Jarak tempuh dari jalan kota ke-1 ke kota ke-3 adalah 15.
- Jarak tempuh dari jalan kota ke-1 ke kota ke-4 adalah 20.
- Jarak tempuh dari jalan kota ke-2 ke kota ke-3 adalah 35.
- Jarak tempuh dari jalan kota ke-2 ke kota ke-4 adalah 25.
- Jarak tempuh dari jalan kota ke-3 ke kota ke-4 adalah 30.

Asumsi lainnya adalah suatu kota tidak memiliki jalan ke kota itu sendiri sehingga apabila dituliskan matriks ketetanggaannya akan bernilai tak hingga.

Apabila digambarkan dalam bentuk graf, maka akan terbentuklah graf seperti berikut ini.



Gambar 1. Graf Lengkap

Untuk memecahkan TSP menggunakan algoritma *branch and bound*, kita perlu untuk membuat sebuah pohon ruang status untuk memudahkan dalam menandakan kota mana saja yang sudah dikunjungi. Karena berawal dari kota ke-1 maka akar dari pohon ruang status kita adalah *node* yang menandakan bahwa kita sudah pernah mengunjungi kota ke-1. Setiap *node* pada pohon ruang status menggambarkan rute perjalanan si *salesman* dan harus dihitung *cost*-nya. Untuk akar pohon ruang status akan didapatkan *cost*-nya bernilai 75.

$$\text{Cost awal} = 0.5*((10+15)+(10+25)+(15+30)+(20+25))$$
$$= 0.5 * 150$$
$$= 75$$

Pada perhitungan diatas, kita harus mencari semua dua busur dengan bobot paling minimum untuk setiap kota atau setiap simpul yang bersisian dengan busur tersebut. Contohnya pada kota ke-1, dua busur dengan bobot paling minimum dan bersisian dengan kota ke-1 ialah busur dengan bobot 10 yang mengarah ke kota ke-2 dan busur dengan bobot 15 yang mengarah ke kota ke-3. Begitu juga pada kota – kota yang lainnya. Pada kota ke-2 dua busur dengan bobot paling minimum adalah busur dengan bobot 10 yang mengarah ke kota ke-1 dan busur dengan bobot 25 yang mengarah ke kota ke-4. Sedangkan pada kota ke-3 dua busur dengan bobot paling kecil adalah busur yang mengarak ke kota ke-4 dengan bobot 30 dan busur yang mengarah ke kota ke-1 dengan bobot 15. Dan yang terakhir pada kota ke-4 dua busur dengan bobot paling minimum adalah busur yang bersisian juga denga kota ke-1 dengan bobot 20 dan busur yang bersisian juga dengan kota ke-2 dengan bobot 25. Bobot semua busur tadi dijumlahkan semua lalu dibagi dua dan itulah yang menjadi *cost* awal, sehingga jika kita hitung akan didapatkan *cost* awalnya bernilai 75.

Dari *node* akar, kita akan mendapatkan tiga cabang. Ketiga cabang ini mewakili setiap kota yang akan dikunjungi *salesman* terebut. Setiap *node* pada pohon ruang status akan dihitung *cost*-nya dan akan dimasukkan ke dalam *priority queue*. Berikut ini perhitungan ketiga *node* dari *node* akar pohon ruang status kita.

$$\textit{Cost node } 12 = 0.5*((\underline{10}+15)+(\underline{10}+25)+(15+30)+(20+25))$$
$$= 0.5 * 150$$
$$= 75$$
$$\textit{Cost node } 13 = 0.5*((10+\underline{15})+(10+25)+(\underline{15}+30)+(20+25))$$
$$= 0.5 * 150$$
$$= 75$$
$$\textit{Cost node } 14 = 0.5*((10+\underline{20})+(10+25)+(15+30)+(\underline{20}+25))$$
$$= 0.5 * 155$$
$$= 77.5$$

Angka – angka diberi garis bawah merupakan jarak tempuh dari jalan yang dipilih oleh *salesman* sehingga harus ditambahkan ke dalam perhitungan cost setiap node. Sedangkan angka yang tidak diberi garis bawah merupakan busur dengan bobot terkecil yang bersisian dengan suatu kota. Setelah dihitung *cost* dari setiap *node*, *node* akan dimasukkan kedalam *priority queue* diurutkan berdasarkan *cost* yang paling kecil nilainya. Maka akan isi *priority queue* yang akan didapatkan seperti dibawah ini.

Node	12	13	14
Cost	75	75	77.5

Tabel 1. *Priority Queue* setelah percabangan *node* akar

Setelah itu kita kembali melakukan percabangan dengan node yang memiliki cost paling kecil. Percabangan pada *node* 12 akan menghasilkan 2 *node* baru, yaitu *node* 123 dan *node* 124. *Node* 123 dan n*ode* 124 akan kembali kita hitung *cost*-nya. Berikut merupakan rincian perhitungan cost dari kedua *node* tersebut.

$$\text{Cost node } 123 = 0.5*((\underline{10}+15)+(\underline{10}+\underline{35})+(15+\underline{35})+(20+25)) = 0.5 * 165 = 82.5$$

$$\text{Cost node } 124 = 0.5*((\underline{10}+15)+(\underline{10}+\underline{25})+(15+30)+(20+\underline{25})) = 0.5 * 150 = 75$$

Seperti sebelumnya, angka – angka yang digarisbawahi merupakan jarak tempuh dari alan yang dilalui oleh si *salesman* dan angka yang tidak digarisbawahi merupakan bobot dari busur yang memiliki bobot paling minimum yang bersisian dengan suatu kota. Kedua node tadi pun kita masukkan lagi ke dalam *priority queue* sehinnga akan didapatkan *priority queue* sebagai berikut.

Node	13	124	14	123
Cost	75	75	77.5	82.5

Tabel 2. *Priority Queue* setelah percabangan *node* 12

Setelah itu kita kembali melakukan percabangan pada *node* dengan *cost* yang paling kecil. Kali ini kita melakukan percabangan pada *node* 13. Percabangan pada *node* 13 ini akan merngahasilkan dua *node* baru yaitu *node* 132 dan *node* 134. Kedua *node* tersebut akan dihitung *cost*-nya masing – masing. Berikut ini merupakan detil perhitungan *cost* dari setiap *node*.

$$\text{Cost node } 132 = 0.5*((10+\underline{15})+(10+\underline{35})+(\underline{15}+\underline{35})+(20+25)) = 0.5 * 165 = 82.5$$

$$\text{Cost node } 134 = 0.5*((10+\underline{15})+(10+25)+(\underline{15}+\underline{30})+(20+\underline{30})) = 0.5 * 155 = 77.5$$

Seperti sebelumnya, angka – angka yang digarisbawahi merupakan jarak tempuh dari alan yang dilalui oleh si *salesman* sehingga harus dihitung pada perhitungan *cost.* Angka yang tidak digarisbawahi adalah bobot minimum dari bususr yang bersisian dengan suatu kota. Pada kota ke-3 kita tidak lagi mencari busur yang bersisian dengan bobot paling kecil karena kota ke-3 sudah ada bobot yang harus diambil. Kedua node tadi pun kita masukkan lagi ke dalam *priority queue* sehinnga akan didapatkan *priority queue* sebagai berikut.

Node	124	14	134	123	132
Cost	75	77.5	77.5	82.5	82.5

Tabel 3. *Priority Queue* setelah percabangan *node* 13

Setelah memasukkan *node*ke dalam *priority queue*, kita kembali melakukan ekspansi pada *node* berdasarkan *cost* yang paling kecil yang ada di dalam *priority* queue. Dapat dilihat pada tabel 3 bahwa kita akan melakukan ekspansi pada *node* 124 karena memiliki *cost* paling kecil dengan nilai 75. Hasil ekspansi dari *node* 124 ini akan menghasilkan hanya 1 *node* yaitu *node* 1243. *Node* 1243 ini merupakan salah satu solusi yang kita dapat saat ini karena pada *node* ini semua kota sudah dikunjungi tepat sekali. Akan tetapi *node* 1243 ini belum tentu menjadi solusi TSP karena mungkin didapatkan solusi lain dengan total jarak tempuh kebih kecil dari *node* 1243. Oleh karena itu percabangan pada *node* – *node* yang ada pada *priority queue* tetap harus dilakukan. Namun percabangan pada *node* – *node* yang ada di *prioritu queue* ini harus memenuhi satu syarat yaitu nilai *cost*-nya tidak lebih besar dari nilai *cost* solusi yang ditemukan saat ini. Apabila nilai *cost*-nya lebih besar dari nilai *cost* solusi saat ini maka *node* tidak akan dilakukan percabangan pada *node* tersebut. Berikut ini merupakan perhitung secara detil *cost* dari *node* 1243.

$$\text{Cost node } 1243 = 0.5*((\underline{10}+\underline{15})+(\underline{10}+\underline{25})+(\underline{15}+\underline{30})+(\underline{30}+\underline{25})) = 0.5 * 160 = 80$$

Karena *node* 1243 sudah mengunjungi semua kota yang ada dan *node* 1243 merupakan solusi sejauh ini, maka node 1243 tidak akan dimasukkan ke dalam *priority queue.* Pada perhitungan cost diatas pun dapat dilihat bahwa semua angka diberi garis bawah yang menandakan jalan yang dilalui oleh salesman. Dan dapata dilihat pada perhitungan diatas tidak ada angka yang tidak digarisbawahi karena si *salesman* tersebut sudah mengunjungi semua kota yang ada sehingga kita tidak harus mencari busur dengan bobot paling minimum. Karena node 1243 tidak dimasukkan ke dalam priority queue, maka priority queue yang kita miliki saat ini adalah sebagai berikut.

Node	14	134	123	132
Cost	77.5	77.5	82.5	82.5

Tabel 4. *Priority Queue* setelah memproses *node* 124

Setelah itu kita kembali lagi untuk mengekspansi *node* yang ada pada *priority queue*. *Node* yang akan kita ekspansi kali ini adalah *node* 14 karena memiliki *cost* paling kecil dan berada pada *head* dari *priority queue*. Dengan mengekspansi *node* 14 kita akan mendapatkan dua *node* yang baru yaitu *node* 142 dan *node* 143. Berikut merpuakan rincian perhitungan dua node tersebut.

$$\text{Cost node } 142 = 0.5*((10+\underline{20})+(10+\underline{25})+(15+30)+(\underline{20}+\underline{25})) = 0.5 * 155 = 77.5$$

$$\textit{Cost node } 143 = 0.5*((10+\underline{20})+(10+25)+(15+\underline{30})+(\underline{20}+\underline{30}))$$
$$= 0.5 * 160$$
$$= 80$$

Pada penjumlahan terakhir (20+25) atau pun (20+30), keduanya diberikan garis bawah yang menandakan jarak tempuh dari jalan yang dilalui *salesman* dari kota ke-4 sehingga tidak perlu mencari bobot busur terendah dari kota ke-4. Lalu seperti biasa kita kembali memasukkan kedua node ini ke dalam priority queue yang kita miliki. Dengan demikian maka kita akan memiliki priority queue dengan isi sebagai berikut.

Node	134	142	143	123	132
Cost	77.5	77.5	80	82.5	82.5

Tabel 5. Priorty Queue setelah percabangan node 14

Lalu kita kembali melanjutkan percabangan *node* yang ada. Dengan melihat tabel *priority queue* diatas, kini kita akan mleanjutkan melakukan percabangan *node* 134 yang berada pada *head* dari *priority queue* kita. Dari percabangan node 134, kita akan mendapatkan satu simpul yaitu node 1342. Node 1342 ini juga merupakan salah satu kandidat dari solusi TSP karena pada node 1342 si salesman sudah mengunjungi semua kota yang ada. Meskipun begitu node 1342 belum tentu menjadi solusi TSP. Node 1342 dapat menjadi solusi TSP apabila memiliki nilai cost paling rendah. Berikut merupakan detil dari perhitungan *cost* pada *node* 134.

$$\textit{Cost node } 1342$$
$$= 0.5*((\underline{10}+\underline{15})+(\underline{10}+\underline{25})+(\underline{15}+\underline{30})+(\underline{25}+\underline{30}))$$
$$= 0.5 * 160$$
$$= 80$$

Semua angka diatas diberi garis bawah yang artinya merupakan jalan yang dilalui oleh salesman sehingga kita tidak perlu mencari busur dengan bobot minimum. Dapat dilihat bahwa hasil perhitungan *cost* dari *node* 1342 adalah 80. *Cost* dari *node* 1342 bernilai sama dengan cost dari *node* 1243 sehingga *node* 1342 dan *node* 1243 merupakan solusi saat ini. Karena *node* 1342 merupakan solusi, maka kita tidak akan memasukkannya ke dalam *priority queue*. *Priority queue* yang kita miliki setelah mengekspansi *node* 134 adalah sebagai berikut.

Node	142	143	123	132
Cost	77.5	80	82.5	82.5

Tabel 6. *Priority Queue* setelah percabangan *node* 134

Setelah mendapatkan dua solusi sejauh ini, kita tidak berhenti untuk mengekspansi node yang mungkin menghasilkan solusi. Oleh karena itu, kita tetap mengekspansi *node* yang ada pada *head priority queue*. *Node* 142 dengan *cost* 77.5 yang akan kita ekspansi saat ini. *Node* 142 ini akan menghasilkan sebuah *node* 1423 yang dapat menjadi solusi lainnya. Namun tergantung pada *cost*-nya apakah solusi yang dihasilkan merupakan solusi yang lebih baik dari sebelumnya atau tidak. Berikut merupakan perhitungan *cost* dari hasil ekspansi *node* 142 secara detil.

$$\textit{Cost node } 1423$$
$$= 0.5*((\underline{10}+\underline{20})+(\underline{10}+\underline{35})+(\underline{35}+\underline{30})+(\underline{20}+\underline{30}))$$
$$= 0.5 * 190$$
$$= 95$$

Melihat dari detil perhitungan diatas, *node* 1423 memiliki *cost* bernilai 90. Walaupun pada *node* 1423 sudah mengunjungi semua kota yang ada, tetapi *cost* yang didapatkan masih lebih besar dari cost sebelumnya yaitu 80. Oleh karena itu, dapat dikatakan bahwa *node* 1423 bukan merupakan soslusi yang optimum dan berarti bukan merupakan solusi TSP. Karena *node* 1423 sudah tidak bisa lagi diekspansi, maka *node* tersebut tidak akan dimasukkan ke dalam *priority queue,* sehingga *priority queue* yang kita miliki saat ini adalah sebagai berikut.

Node	143	123	132
Cost	80	82.5	82.5

Tabel 7. Priority Queue setelah mengekspansi node 143

Setelah itu kita kembali mengekspansi *node* yang masih layak untuk diekspansi. *Node* selanjutnya yaitu *node* 143. Karena node 143 memiliki cost 80 dan cost tersebut tidak lebih besar dari solusi yang kita miliki saat ini maka *node* 143 masih layak untuk diekspansi dan tetap harus kita ekspansi. Dengan mengekspasi *node* 143 kita, akan mendapatkan satu solusi lainnya. Namun sama seperti sebelumnya, hasil ekspansi dari *node* 143 belum tentu hasil yang optimal karena *cost* yang dihasilkan belum tentu lebih kecil dari *cost* solusi saat ini. Berikut ini merupakan perhitungan *node* 1432 yang merupakan hasil ekspansi dari *node* 143.

$$\textit{Cost node } 1432$$
$$= 0.5*((\underline{10}+\underline{20})+(\underline{10}+\underline{35})+(\underline{35}+\underline{30})+(\underline{20}+\underline{30}))$$
$$= 0.5 * 190$$
$$= 95$$

Dari hasil perhitungan diatas, dapat kita tarik sebuah kesimpulan bahwa node 1432 bukan merupakan solusi yang optimal dikarenakan nilai *cost*-nya yang melebihi 80 atau cost dari solusi terbaik saat ini. Karena *node* 1432 merupakan salah satu solusi, walaupun tidak optimal, dan tidak dapat diekspansi lagi, maka *node* 1432 tidak akan dimasukkan ke dalam *priority queue*. Sehingga setelah ekspansi node 143 dilakukan maka kita akan mendapatakan priority queue yang berisi sebagai berikut.

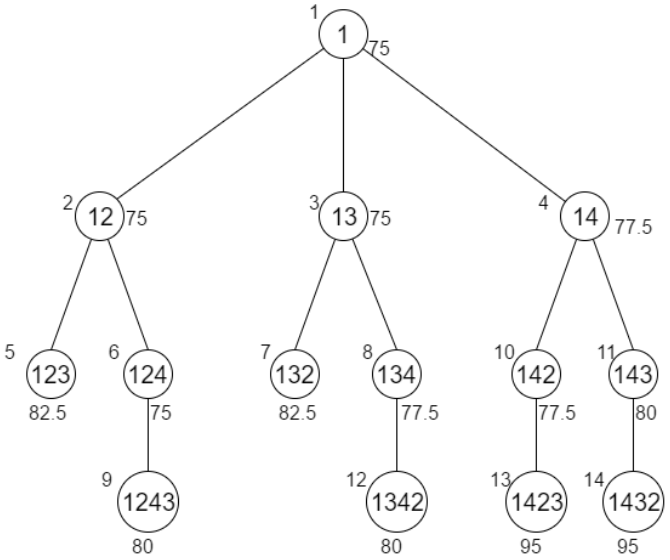
Node	123	132
Cost	82.5	82.5

Tabel 8. Priority Queue setelah percabangan pada node 143

Setelah itu kita kembali melanjutkan ekspansi *node* yang ada dalam *priority queue*. *Node* yang akan diekspansi tentu

node yang masih layak atau yang memiliki *cost* kurang dari sama dengan *cost* dari sousi terbaik saat ini yaitu 80. Dengan melihat tabel *priority queue* pada tabel 8, dapat kita lihat bahwa tidak ada lagi *node* yang layak untuk diekspansi karena semua node yang ada dalam *priority queue* memiliki *cost* yang lebih besar dari *cost* solusi terbaik saat ini. Oleh karena itu kita tidak perlu melanjutkan ekspansi pada node yang tersisa dan mengakhiri pencarian solusi lainnya.

Jika kita membuat pohon ruang status makan kita akn mendapatkan pohon dengan bentuk sebagai berikut.



Gambar 2. Pohon ruang status dari pemecahan TSP

Seperti yang kita lihat bahwa terdapat 14 *node* yang dibangkitkan. Angka yang berada dia sebelah kiri atas *node* merupakan urutan dibangkitkannya *node* tersebut. Sedangkan angka yang ada disebelah kanan ata bawah dari *node* merupakan *cost*-nya. Dapat dilihat dari pohon ruang statusnya nahwa kita mendapatkan 4 solusi dari node – node yang kita ekspansi. Akan tetapi dua dari node tersebut bukanlah solusi yang optimal karen memiliki bobot yang lebih besar dari kedua node lainnya. Karena *node* 1432 dan *node* 1423 memiliki bobot 90 dan terdapat node lain yang memiliki bobot lebih kecil dari 90 maka *node* 1432 dan *node* 1423 bukan merupakan solusi yang optimal. Sedangkan pada *node* 1342 dan *node* 1243, tidak ada *node* lain yang merupakan solusi dan memiliki *cost* lebih kecil dari 80 sehingga kedua *node* tersebut dapat dikatakan menjadi solusi dari permaslaha TSP.

Seperti yang disebutkan sebelumnya bahwa setiap *node* menggambarkan rute yang dilalui oleh *salesman*, maka terdapat dua rute yang dapat dilalui *salesman*. Rute pertama yaitu berawal dari kota ke-1 lalu menuju ke kota ke-2. Setelah itu melanjutkan perjalanan dari kota ke-2 ke kota ke-4 dan dilaknjutkan ke kota ke-3. Dari kota ke-3 *salesman* dapat kembali ke kota asal yaitu kota ke-1. Sedangkan rute kedua memiliki arah kebalikannya. Dimulai dari kota ke-1 salesman pergi menuju ke kota ke-3. Lalu dilanjutkan dengan mngunjungi kota ke-4 dan kota ke-2. Dari kota ke-2 ini *salesman* dapat kembali kemkota awal yaitu kota ke-1. Kedua rute ini memiliki total jarak tempuh yang sama dengan nilai 80.

B. Pemecahan TSP dengan Reduced Cost Matrix

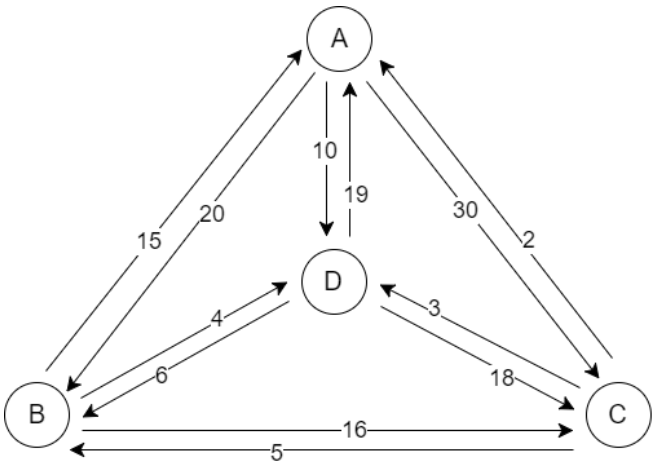
Suatu permasalahan TSP yang mengharuskan kita untuk mecari rute dengan total jarak tempuh terpendek dengan tambahan syarat yaitu harus kembali lagi ke kota awal tempat kita memulai perjalanan. Berikut deskripsi persoalan TSP secara lenoh detil

Seorang *salesman* mamiliki tugas dari tempat kerjanya bahwa ia harus pergi ke beberapa kota untuk memasarkan suatu barang. Ia ditugaskan untuk mengunjungi 4 kota diantranya kota A, kota B, kota C, dan kota D. S*alesman* tersebut harus kembali ke kota awalnya yaitu kota A dengan syarat sudah mengunjungi semua kota tepat satu kali.

Jarak yang harus ditempuh dari suatu kota ke kota lainnya berbeda - beda. Berikut ini merupakan rincian jarak yang harus ditempuh oleh *salesman* dari satu kota ke kota lainnya

- Jarak antara kota A dan kota B adalah 20
- Jarak antara kota A dan kota C adalah 30
- Jarak antara kota A dan kota D adalah 10
- Jarak antara kota B dan kota A adalah 15
- Jarak antara kota B dan kota C adalah 16
- Jarak antara kota B dan kota D adalah 4
- Jarak antara kota C dan kota A adalah 2
- Jarak antara kota C dan kota B adalah 5
- Jarak antara kota C dan kota D adalah 3
- Jarak antara kota D dan kota A adalah 19
- Jarak antara kota D dan kota B adalah 6
- Jarak antara kota D dan kota C adalah 18

Jika kita modelkan menggunakan graf dengan busur merepresentasikan sebagai jalan dengan bobot sebagai jarak yang harus ditempuh dan simpul merepresentasikan sebuah kota maka kita akan mendapatkan graf seperti berikut ini.



Gambar 3. Graf lengkap

Sesuai dengan nama cara untuk menentukan *cost*-nya, kita harus terlebih dahulu membuat matriks ketetanggaannya . Berikut ini merupakan matriks ketetanggan dari graf pada gambar 3.

$$\begin{bmatrix} \infty & 20 & 30 & 10 \\ 15 & \infty & 16 & 4 \\ 2 & 5 & \infty & 3 \\ 19 & 6 & 18 & \infty \end{bmatrix}$$

Untuk memecahkan TSP menggunakan algoritma branch and bound, kita harus menginisiasi simpul akar dari pohon ruang status. Karena kita harus memulai dari kota A maka simpul akar kita akan menandakan bahwa kita sudah mengunjungi kota A. Sama seperti pada saat memecahkan TSP menggunakan bobot tur lengkap, kita juga harus menentukan *cost* awal atau *cost* dari simpul akar dari pohon ruang status. Penentuan *cost* ini kita lakukan dengan cara mereduksi matriks ketetanggan. Berikut ini merupakan hasil dari reduksi matriks pada simpul akar dan perhitungan *cost*-nya.

$$\begin{bmatrix} \infty & 10 & 8 & 0 \\ 11 & \infty & 0 & 0 \\ 0 & 3 & \infty & 1 \\ 13 & 0 & 0 & \infty \end{bmatrix}$$

Cost simpul awal = 10 + 4 + 2 + 6 + 12
= 34

Untuk menghitung cost pada reduced cost matrix, kita perlu membentuk tiap baris dan kolom pada matriks memiliki nilai 0 dengan cara mengurangi dengan nilai yang paling kecil dari tiap baris terlebih dahulu. Angka 10, 4, 2 dan 6 merupakan nilai terkecil dari setiap baris. Setelah mengurangi setiap baris dengan angka terkecil hanya ada satu kolom yang masih tidak memiliki nilai 0 yaitu kolom ke-3. Sehingga untuk membuat kolom ke-3 memiliki setidaknya satu nilai 0 kita harus mengurangi setiap elemen pada kolom ke-3 dengan nilai paling kecil dari kolom ke-3 yaitu 12. Pada perhitungan *cost* awal semua nilai yang dibutuhkan untuk mereduksi matriks akan dijumlahkan dan akan dijadikan sebagai *cost* simpul akar.

Setelah itu kita lanjutkan dengan mengekpansi simpul akar. Hasil dari ekspansi simpul akar akan mengasilkan tiga simpul baru. Setiap simpul memiliki *cost* dan matriks tersendiri. Berikut merupakan hasil reduksi matriks dan perhitungan dari ketiga simpul tersebut.

$$Node\ AB \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 0 \\ 0 & \infty & \infty & 1 \\ 13 & \infty & 0 & \infty \end{bmatrix}$$

Cost simpul AB = 34 + 0 + 10
= 44

$$Node\ AC \begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & 0 \\ \infty & 2 & \infty & 0 \\ 2 & 0 & \infty & \infty \end{bmatrix}$$

Cost simpul AC = 34 + 8 + 1 + 11
= 54

$$Node\ AD \begin{bmatrix} \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty \\ 0 & 3 & \infty & \infty \\ \infty & 0 & 0 & \infty \end{bmatrix}$$

Cost simpul AD = 34 + 0 + 0
= 34

Seperti yang kita ketahui untuk mereduksi melanjutkan mereduksi matrkis kita harus menentukan kota awal dan kota tujuan. Simpul AB menandakan sang *salesman* berjalan dari kota A ke kota B dan begitu juga dengan simpul lainnya. Kota awal dan kota tujuan ini penting untuk mengetahui baris dan kolom mana yang setiap elemennya harus kita ganti nilainya menjadi tak hingga. Contohnya pada simpul AD. Kita harus membuat setiap elemen pada baris ke-1 memiliki nilai tak hingga karena kota A merupakan kota awal. Lalu kita juga harus membuat semua nilai pada kolom ke-4 memiliki nilai tak hingga karena kota tujuan adalah kota D. Elemen pada kolom ke-1 dan baris ke-4 juga harus diganti nilainya menjadi tak hingga juga. Mengganti elemen pada kolom berdasarkan kota asal dan baris berdasarkan kota tujuan bertujuan agar tidak memilih jalur yang membentuk sirkular.

Untuk perhitungan cost setiap simpul dilakukan dengan cara menjumlahkan cost pada simpul sebelumnya, nilai yang dibutuhkan untuk merduksi matriks, dan nilai dari kolom berdasaran kota tujuan dan baris berdasarkan kota asal dari matriks pada simpul sebelumnya. Contohnya pada simpul AC misalnya. Setelah membuat setiap elemen pada baris ke-1 dan kolom ke-3 menjadi tak hingga dan juha membuat satu elemen pada kolom ke-1 dan baris ke-3 bernilai tak hingga, akan ada baris dan kolom yang tidak memiliki nilai 0. Untuk membuat setiap baris dan kolom maka matriks harus direduksi lagi dengan cara mengurangi dengan nilai paling kecil sampai setiap baris dan kolom emiliki setidaknya satu nilai 0. Pada kasus simpul AC, kolom ke-1 dan baris ke-3 masih belum memiliki nilai 0. Oleh karena itu matriks pada simpul AC, khususnya pada kolom ke-1 akan dikurangi dengan 11 sedangkan pada baris ke-3 dikurangi dengan 1. Lalu angka yang dibutuhkan untuk mereduksi matriks pada simpul AC akan dijumlahkan dengan *cost* dari simpul *parent*-nya. Tidak hanya dengan *cost* dari sumpul parent-nya tetapi juga ditambahkan nilai dari elemen pada baris ke-1 dan kolom ke-3 pada matriks pada simpul *parent*-nya. Itulah mengapa cost pada simpul AC bernilai 54.

Setelah itu semua simpul yang dihasilkan akan dimasukkan ke dalam *priority queue* yang akan mengurutkan smpul mana yang harus kita ekspansi berdasarkan cost terendah.. Karena sebelumnya priority queue yang kita miliki kosong maka saat ini *priority queue* kita memiliki tiga simpul di dalamnya. Berikut merupakan tabel *priority queue*.

Node	AD	AB	AC
Cost	34	44	54

Tabel 9. *Priority Queue* setelah ekspansi simpul A

Setelah itu kita kembali melakukan percabangan pada simpul dengan *cost* paling minimum. Pada percabangan simpul AD, kita akan mendapatkan dua simpul baru yaitu ADB dan ADC. Sama seperti sebelumya kita harus membuat matriks

tereduksi dan menghitung *cost*-nya untuk kedua simpul tersebut. Berikut ini merupakan hasil reduksi matriks dan perhitungan cost secara detil dari hasil ekspansi pada simpul AD.

$$Node\ ADB \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty \\ 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix}$$

Cost simpul ADB = 34 + 0 + 0
= 34

$$Node\ ADC \begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix}$$

Cost simpul ADC = 34 + 0 + 11 + 3
= 48

Pada saat melakukan reduksi matrik untuk simpul ADC, akan didapatkan bahwa baris ke-2 dan baris ke-3 belum memiliki nilai 0 di salah satu elemennya. Oleh karena itu setiap elemen pada baris k-2 harus dikurangi dengan elemen terkecil pada baris tersebut (11) dan begitu pula pada baris ke-3 yang harus mengurani setiap elemennya dengan nilai terkecil pada baris tersebut (3). Berbeda pada saat mereduksi matriks untuk simpul ADB. Pada saat melakukan reduksi matriks untuk simpul ADB kita tidak menemukan baris atau pun kolom yang tidak memiliki nilai 0 di salah satu elemen pada setiap baris dan setiap kolom, tentu kolom atau baris yang semua elemennya bernilai tak hingga adalah pengecualian. Sehingga pada simpul ADB kita tidak perlu mengurangi elemen dari matriks dan membuat cost-nya lebih rendah.

Setelah membuat matriks dan menghitung *cost*-nya, kini kita kembali memasukkan kedua simpul tersebut ke dalam *priority queue*. Berikut ini merupakan tabel *priority queue* saat ini.

Node	ADB	AB	ADC	AC
Cost	34	44	48	54

Tabel 10. Priority Queue setelah ekspansi simpul AD

Lalu kita kembali melanjutkan ekspansi pada simpul dengan *cost* paling kecil. Ekspansi kali ini dilakukan pada simpul ADB. Ekspansi pada simpul ADB ini hanya akan menghasilkan satu cabang dan hasil dari ekspansi ini merupakan solusi pertama yang kita dapat. Berikut merupakan matriks tereduksi dan cost dari simpul ADBC.

$$Node\ ADBC \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix}$$

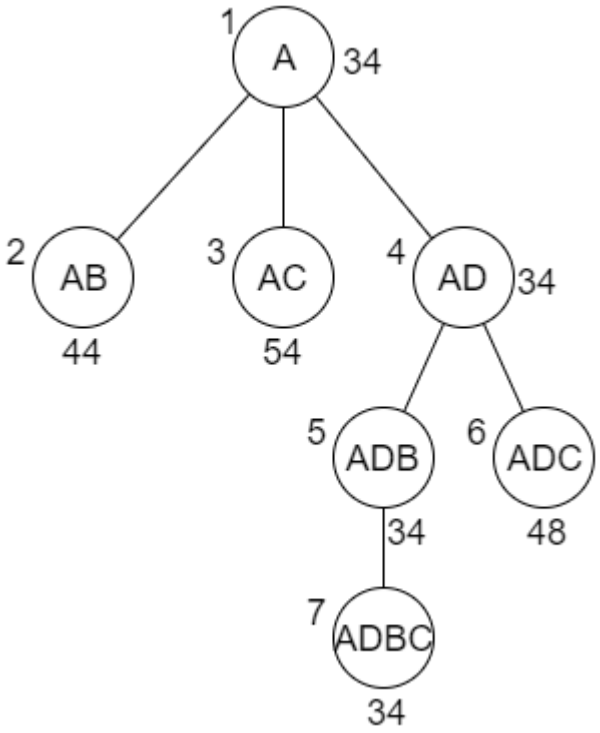
Dapat kita lihat bahwa semua elemen dari matriks tereduksi pada simpul ADBC bernilai tak hingga. Ini dikarenakan semua kota yang ada sudah dikunjungi pada simpul ADBC dengan rute A sebagai kota pertama yang dikunjungi, D sebagai kota kedua, B sebagai kota ketiga, dan C sebagai kota yang terakhir dikunjungi *salesman* sebelum kembali ke kota awal. Simpul ADBC ini merupakan simpul daun sehingga cara mengitung cost-nya ialah dengan menjumlah langsung jarak – jarak yang dtempuh salesman sesuai dengan rute yang didapat. Karena simpul ADBC merupakan solusi, maka kita tidak perlu memasukkan ke dalam *priority queue*. Berikut merupakan *priority queue* yang kita miliki saat ini.

Node	AB	ADC	AC
Cost	44	48	54

Tabel 11. *Prority Queue* setelah ekspansi simpul ADB

Karena kita sudah mendapatkan sebuah solusi, maka kita hanya akan erlu mengekspansi simpul dengan *cost* yang tidak lebih besar dari *cost* solusi yang kita miliki. Namun dapat dilihat pada tabel diatas bahwa tidak ada lagi simpul yang layak untuk diekspansi karena semua simpul yang ada memiliki *cost* yang lebih besat dari *cost* solusi kita saat ini yaitu 34. Sehingga percabangan atau ekspansi tidak akan dilanjutkan dan simpul ADBC meruakan satu – satunya solusi yang kita dapatkan.

Apabila digambarkan pohon ruang statusnya dari awal maka kita akan mendapatkan seperti gambar berikut.

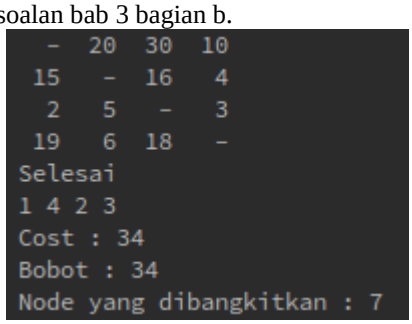


Gambar 4. Pohon ruang status

Angka yang ada pada kira atas simpul merupaka urutan pembangkitan simpul tersebut. Sedangkan angka pada kanan atau bawah simpul merupakan *cost* dari simpul tersebut.

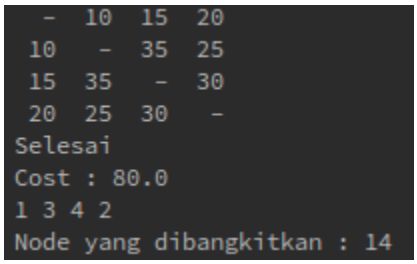
C. Hasil Eksekusi Program

Berikut ini merupakan hasil dari eksekusi program algoritma *branch and bound* dengan perhitungan *cost* menggunakan cara *reduced cost matrix* dalam menyelesaikan TSP pada persoalan bab 3 bagian b.



Gambar 5. Hasil eksekusi program algoritma *branch and bound* menggunakan *reduced cost matrix*

Berikut ini merupakan hasil dari eksekusi program algoritma *branch and bound* dengan perhtungan *cost* menggunakan cara bobot tur lengkap dalam menyelesaikan TSP pada persoalan bab 3 bagian a.



Gambar 6. Hasil eksekusi program algoritma *brach and bound* menggunakan bobot tur lengkap

Dapat kita lihat bahwa persoalan TSP dapat kita selesaikan baik menggunakan *reduced cost matrix* ataupun bobot tur lengkap. Hasil eksekusi program hanya akan menampilkan satu jawaban saja.

IV. KESIMPULAN

Melihat dari dua contoh yang ada pada bab 3, dapat kita simpulkan bahwa algoritma branch and bound dapat menyelesaikan persoalan TSP dengan baik. Perbedaan cara dalam penentuan *cost*-nya pun tidak masalah karena keduanya dapat menghasilkan hasil yang optimum. Untuk graf tidak berarah, menggunakan cara bobot tur lengkap merupakan cara yng paling mudah. Namun untuk graf berarah menggunakan *reduced cost matrix* akan lebih baik.

REFERENSI

[1] https://karyatulisilmiah.com/pengertian-optimasi-dan-linear-programming-2/ diakses pada tanggal 16 Mei 2017 pk. 08.18 WIB

[2] http://industrialengineeringdepartment.blogspot.co.id/2015/04/travelling-salesman-problem-tsp.html diakses pada tanggal 16 Mei 2017 pk. 13.11 WIB

[3] Kelvin. Albertus, 2016, "Aplikasi Program Dinamis dalam Pemecahan TSP", Makalah IF2122 Strategi Algoritma – Sem.II Tahun 2015/2016, Bandung: Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung.

[4] Tim Pengajar IF2211, 2016, "Branch and Bound", Bandung : Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung.

[5] Munir. Rinaldi, 2009, "Diktat IF2211 Strategi Algoritma", Bandung : Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Mei 2017

Irfan Ariq - 13515112