Analisis Pengimplementasian Algoritma Greedy untuk Memilih Rute Angkutan Umum

Arieza Nadya -- 13512017
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
ariezanadya@s.itb.ac.id

***Abstrak*—Semakin dengan berkembangnya zaman, semakin banyak dan padat pula kebutuhan manusia. Oleh karena bertambahnya kebutuhkan manusia ini, orang membutuhkan cara agar pekerjaan-pekerjaan mereka diselesaikan secara cepat serta efisien. Dengan kebutuhan ini, orang pun mencari dan menemukan berbagai cara untuk mencapai tujuan itu, semakin mendekati instan semakin baik. Masalah yang dibahas dalam makalah ini berupa analisis penggunaan algoritma dalam pengimplementasiannya pada masalah sehari-hari. Analisis dalam makalah ini bersifat umum dan dapat dikembangkan atau dimodifikasi sesuai kebutuhan.**

***Kata Kunci*—greedy, efisiensi, angkutan umum, transportasi**

I. PENDAHULUAN

A. Latar Belakang

Topik untuk makalah ini diambil dari pengalaman penulis dan pengambilan keputusan sederhana pada kehidupan sehari-hari. Pada kasus topik permasalahan yang diambil untuk pembuatan makalah ini, kita diberi dengan berbagai macam pilihan rute-rute angkutan umum. Sedangkan dari pilihan-pilihan tersebut kita dapat memilih satu atau beberapa yang paling sesuai dengan batasan-batasan yang kita inginkan. Penggunaan angkutan umum sudah menjadi hal biasa bagi masyarakat dan nyaris setiap hari digunakan. Bahkan orang yang memilih transportasi pribadi pun masih terkadang menggunakan pilihan angkutan umum karena sesuatu dan lain hal. Selain itu, pengambilan keputusan rute angkutan umum merupakan bentuk keputusan yang paling sering dipertimbangkan para komuter, sehingga makalah ini akan menganalisis keputusan yang didapat dengan dua algoritma *Greedy*, yang pertama dengan faktor biaya dan yang kedua dengan faktor rute. Pada kehidupan sehari-hari, yang menjadi batasan umum kita dalam memilih suatu opsi angkutan umum adalah biaya dan waktu, serta kadang kita juga mempertimbangkan faktor panjang perjalanan. Pada keadaan sebenarnya, faktor waktu serta panjang perjalanan dapat jadi tidak linear, namun pada makalah ini untuk menyederhanakan masalah maka kedua faktor tersebut dapat disatukan dalam faktor rute (lama perjalanan).

B. Tujuan

Tujuan dibuatnya makalah ini adalah untuk menyelesaikan salah satu tugas matakuliah IF2211 Strategi Algoritma berupa makalah serta untuk menganalisis penggunaan pendekatan dua algoritma *Greedy* dalam pengambilan keputusan rute angkutan umum, *greedy* rute (pengambilan rute tercepat/terpendek) dan *greedy* biaya (pengambilan rute/angkutan umum dengan biaya terkecil).

II. TEORI DASAR

A. Prinsip Algoritma Greedy

Prinsip dari algoritma *Greedy* kurang lebih sesuai dengan namanya -- *greedy* (rakus). Dengan algoritma ini, pemakai menyelesaikan suatu permasalahan dengan mengambil langkah berikutnya yang paling menguntungkan dari pilihan-pilihan yang ada pada langkah tersebut dan mengambilnya. Pada algoritma *greedy*, setelah mengambil keputusan pada suatu langkah, maka tidak dapat lagi kembali ke langkah-langkah yang sebelumnya. Pada setiap langkah, algoritma memilih pilihan paling optimum. Pada langkah tersebut, pilihan yang dipilih merupakan pilihan optimum lokal. Harapan yang ingin dicapai dengan pemilihan sebuah optimum lokal adalah sisa dari langkah-langkahnya akan mencapai sebuah optimum global pada langkah terakhir. Karena prinsip tersebutlah maka algoritma ini disebut dengan algoritma rakus.

Elemen-elemen yang terdapat pada algoritma *Greedy* adalah sebagai berikut:

1. Himpunan kandidat, *C*
   Himpunan yang berisi kandidat elemen-elemen yang dapat membentuk solusi.

2. Himpunan solusi, *S*
   Himpuan yang berisi elemen-elemen solusi

3. Fungsi seleksi (*selection function*).
   Fungsi seleksi adalah sebuah fungsi yang membatasi serta memilih kandidat terbaik untuk

dijadikan himpunan solusi.

4. Fungsi kelayakan (*feasible*)
   Fungsi kelayakan menentukan apakah kandidat dapat memberikan solusi

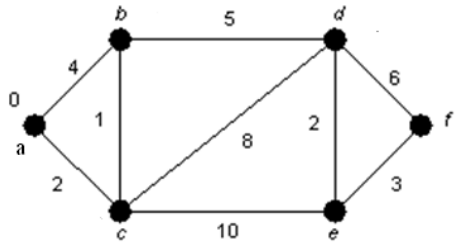
5. Fungsi solusi
   Fungsi solusi memberikan sebuah nilai true apabila solusi sudah ditemukan dan false apabila solusi belum ditemukan.

6. Fungsi obyektif
   Fungsi obyektif mengindikasikan apabila solusi sudah lengkap atau belum lengkap.

Dengan elemen-elemen tersebut, cara kerja algoritma *Greedy* yaitu melibatkan pencarian sebuah himpunan bagian, *S*, dari himpunan kandidat, *C*. Himpunan solusi *S* harus memenuhi beberapa kriteria, yaitu menyatakan suatu solusi dan *S* dioptimisasi oleh fungsi obyektif.

Pada algoritma *Greedy* sebuah solusi optimum global belum tentu merupakan solusi terbaik, namun merupakan *sub-optimum* atau *pseudo-optimum*. Hal ini terjadi karena algoritma *greedy* tidak bekerja secara keseluruhan, karena hanya mengambil satu *path*, sedangkan kemungkinan lainnya tidak dipertimbangkan. Selain itu juga terdapat fungsi seleksi yang berbeda. Untuk mencapai sebuah solusi optimal harus digunakan sebuah fungsi seleksi yang tepat. Karena itu, pemakaian algoritma *greedy* tidak selalu berhasil memberikan solusi optimal, namun untuk permasalahan yang tidak harus memberikan jawaban terbaik mutlak, maka algoritma *greedy* dapat digunakan untuk memberikan sebuah solusi hampiran (*approximation*). Pada masalah seperti itu, penggunaan algoritma *greedy* lebih baik daripada menggunakan algoritma lain yang lebih rumit untuk memberikan solusi eksak.

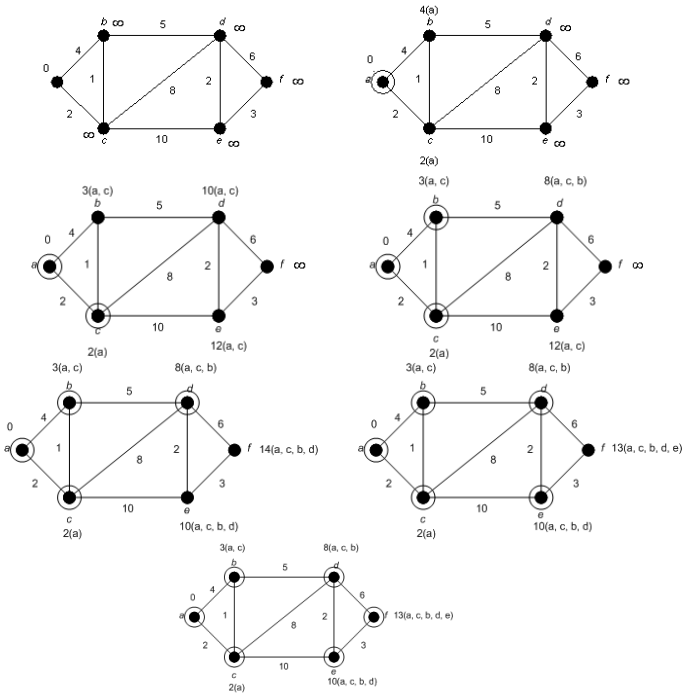
Berikut dibahas contoh penggunaan algoritma *greedy* dalam persoalan *shortest path* dengan algoritma Dijkstra., graf tertera pada Gambar 2.1.



Gambar 2.1 Graf G[1]

Pada gambar di atas, graf G menggambarkan rute-rute yang akan dipilih. Lintasan terpendek dari sebuah node ke node yang lain dapat dicari dengan strategi *greedy* sebagai berikut:

- Pada setiap langkah, ambil sisi yang berbobot minimum yang menghubungkan sebuah simpul yang sudah terpilih dengan simpul lain yang belum terpilih.
- Lintasan dari simpul asal ke simpul yang baru haruslah merupakan lintasan yang terpendek di antara semua lintasannya ke simpul-simpul yang belum terpilih.



Gambar 2.2 Visualisasi penyelesaian *shortest path* dari node a ke node f menggunakan *greedy*[1]

B. Prinsip Algoritma Branch-and-Bound

Algoritma *branch and bound* merupakan perbaikan dari algoritma BFS (*Breadth First Search*). Perbedaan dari algoritma *Branch and Bound* dengan BFS adalah: BFS murni mengekspansi simpul berdasarkan urutan pembangkitan. Sedangkan algoritma *branch and bound* memberi setiap simpul dengan suatu nilai cost (berupa taksiran lintasan termurah dari simpul awal ke simpul tujuan). Simpul-simpul dibangkitkan berdasarkan cost yang terkecil.

Pada umumnya untuk banyak masalah, simpul solusi tidak diketahui letaknya. Maka untuk menentukan cost dari simpul-simpul yang ada digunakan taksiran dengan fungsi:

$$\hat{c}(i) = f(i) + \hat{g}(i)$$

c(i) = cost simpul *i*
f(i) = cost mencapai simpul *i* dari akar
g(i) = cost simpul *i* ke simpul tujuan

Algoritma *branch and bound* ini tidak akan membangkitkan simpul yang telah *bounded* (dibatasi) jika simpul tersebut tidak mengarah kepada sebuah simpul solusi.

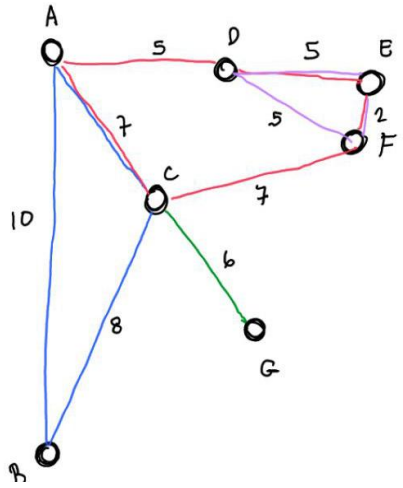
III. ALGORITMA GREEDY PADA PEMILIHAN KEPUTUSAN RUTE ANGKUTAN UMUM

A. Greedy Rute

Pada pemilihan berdasarkan rute atau panjang perjalanan, algoritma tidak memikirkan berapa biaya yang harus dikeluarkan untuk melewati rute tersebut. Algoritma *greedy* rute cukup sederhana dan bekerja seperti algoritma *greedy* untuk menyelesaikan permasalahan *shortest path*.

Algoritma *greedy* rute mencari lintasan terpendek dari simpul awal ke simpul berikutnya.

Contoh lintasan yang digunakan adalah seperti yang tertera pada gambar **Gambar 3.1** berikut:



Gambar G3.1 Graf G

Graf G serta panjang lintasannya disimpan dalam bentuk sebuah matriks dengan isi matriks merupakan panjang dari lintasannya. Untuk Graf G di atas, matriksnya adalah seperti berikut:

	A	B	C	D	E	F	G
A	∞	10	7	5	∞	∞	∞
B	10	∞	8	∞	∞	∞	∞
C	7	8	∞	∞	∞	7	6
D	5	∞	∞	∞	5	5	∞
E	∞	∞	∞	5	∞	2	∞
F	∞	∞	7	5	2	∞	∞
G	∞	∞	6	∞	∞	∞	∞

Tabel T3.1 Matriks Rute G

Matriks pada T3.1 menyatakan panjang lintasan dari sebuah simpul ke simpul yang lain. Untuk simpul yang mengarah ke dirinya sendiri dan simpul yang tidak memiliki lintasan ke sebuah simpul lain diberi dengan tanda ∞.

Pseudo code untuk algoritma *greedy* rute adalah sebagai berikut:

```
procedure greedyroute (input a: integer,
b: integer)

      deklarasi:
      S : array

      algoritma:
      S <-- {} {menginisialisasi himpunan
      solusi        dengan kosong}
      while (b belum ada di S) {
         hidupkan simpul a
         masukkan simpul a ke himpunan
             solusi
         do {
         if (matriks[a][nextsimpul] =/= ∞
         and nextsimpul belum hidup)    {
             hidupkan simpul selanjutnya
         } else { do nothing }
} while (simpul belum habis)

     Pilih simpul lintasan terpendek
     Masukkan simpul ke S
     a <-- simpul current
}
```

Contoh kasus 1:

Rute A ke F

Pada lintasan G kita akan mencari lintasan terpendek dari simpul A ke simpul F. Menggunakan algoritma *greedy* rute, berikut alur pemilihan rutenya:

- Simpul A hidup
- Simpul A menghidupkan ketiga simpul yang terhubung dengan simpul tsb, yaitu simpul B, C, dan D.
- Memasukkan simpul A ke dalam himpunan solusi.
- Memilih simpul dengan lintasan terpendek dari ketiga simpul yang dihidupkan tadi, dalam kasus ini adalah D dengan panjang lintasan 5.
- Memasukkan simpul D ke dalam himpunan solusi.
- Menghidupkan simpul E dan simpul F
- Mengambil simpul E dan memasukkan ke dalam himpunan solusi
- Memasukkan simpul F ke dalam himpunan solusi
- Himpunan solusi selesai terbentuk

Pada pemecahan masalah di atas, rute yang terbentuk ada A — D — E — F dengan panjang lintasan (5+5+2) = 12. Terlihat bahwa solusi tidak memberikan solusi optimal. Pada kasus ini hal ini disebabkan karena jika ada dua simpul yang panjangnya sama, maka algoritma akan memilih simpul yang lebih dahulu dihidupkan.

Contoh kasus 2:

Rute : E ke G

Dengan menggunakan alur yang sama berikut proses pemilihan jalur untuk rute E sampai G:

- Simpul E hidup, masukkan ke himpunan solusi
- Menghidupkan simpul D dan simpul F
- Memilih simpul F
- Memasukkan simpul F ke dalam himpunan solusi
- Menghidupkan simpul C dan simpul D

- Memilih simpul D
- Memasukkan simpul D ke dalam himpunan solusi
- Menghidupkan simpul A
- Memilih simpul A
- Memasukkan simpul A ke dalam himpunan solusi
- Menghidupkan simpul B dan simpul C
- Memilih simpul C
- Memasukkan simpul C ke dalam himpunan solusi
- Menghidupkan simpul G
- Memilih simpul G
- Memasukkan simpul G ke dalam himpunan solusi
- simpul G sudah terdapat di himpunan solusi, selesai.

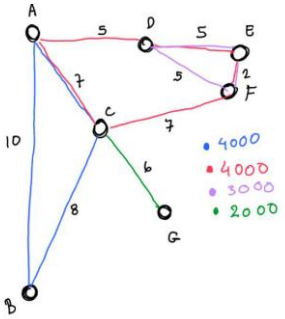
Rute yang dibuat dari E ke G dengan algoritma *greedy* rute adalah sebagai berikut: E - F - D - A - C - G dengan panjang lintasan total (2 + 5 + 5 + 7 + 6) = 25. Solusi yang diberikan juga tidak optimal. Solusi yang lebih pendek (E - F - C - G, dengan panjang lintasan 15) tidak dipertimbangkan pada algoritma *greedy*.

Contoh kasus 3:
Rute D ke C
- Menghidupkan simpul D dan memasukkan simpul D ke dalam himpunan solusi
- Menghidupkan simpul A, E, F
- Memilih simpul A
- Memasukkan simpul A ke dalam himpunan solusi
- Menghidupkan simpul B dan C
- Memilih simpul C
- Memasukkan simpul C ke dalam himpunan solusi,
- Simpul C sudah terdapat dalam himpunan solusi

Pada contoh kasus ini rute yang diambil sudah merupakan solusi optimal.

B. Greedy biaya

Pada algoritma *greedy* biaya, faktor yang diperhitungkan adalah biaya terkecil yang harus dikeluarkan. Algoritma *greedy* biaya dapat jadi memperpanjang rute dari suatu simpul ke simpul lainnya. Asumsi arah perjalanan pada ilustrasi berikut adalah searah jarum jam. Jika yang dicari adalah dari simpul B ke simpul C maka perjalanan yang ditempuh dengan jalur berwarna biru adalah B – A – C. Ilustrasi graf lintasannya beserta biaya yang harus dikeluarkan seperti berikut:



Gambar G3.2 Graf H

Penjelasan pewarnaan pada graf adalah sebagai berikut:
Garis biru: transportasi I
Garis merah : transportasi J
Garis ungu : transportasi K
Garis hijau : transportasi L

Pada algoritma *greedy* biaya, bukan panjang lintasan yang disimpan pada matriks, melainkan transportasi yang melewati suatu simpul disimpan nilainya pada suatu matriks. Selain itu data yang tersimpan juga seperti biaya yang dikeluarkan untuk memakai transportasi tersebut. Selain itu matriks keterhubungan juga diperlukan untuk mengecek jika dua tempat memiliki jalan ke satu sama lain. Jika ada jalan yang menghubungkan kedua tempat, maka ditandai dengan 1, jika tidak ada jalur yang menghubungkan, ditandai dengan 0. Berikut matriks yang dipakai untuk algoritma *greedy* biaya seperti tertera pada Tabel T3.2 dan Tabel T3.3

	A	B	C	D	E	F	G
I	1	1	1	0	0	0	0
J	1	0	1	1	1	1	0
K	0	0	0	1	1	1	0
L	0	0	1	0	0	0	1

Tabel T3.2 Matriks Rute Transportasi

	A	B	C	D	E	F	G
A	0	1	1	1	0	0	0
B	1	0	1	0	0	0	0
C	1	1	0	0	0	1	1
D	1	0	0	0	1	1	0
E	0	0	0	1	0	1	0
F	0	0	1	1	1	0	0
G	0	0	1	0	0	0	0

Tabel T3.3 Matriks keterhubungan

Pada matriks tersebut diberikan informasi jika transportasi melewati suatu simpul atau tidak. Transportasi yang melewati suatu simpul akan ditandai dengan angka 1 dan transportasi yang tidak melewati simpul ditandai dengan angka 0. Pengecekan rute dilakukan secara rekursif untuk mendapatkan rute termurah hingga simpul tujuan.

Pseudo code dari algoritma *greedy* biaya adalah sebagai berikut:

```
  procedure greedycost (input a : integer,
b :    integer)

     deklarasi:
     S : array {himpunan solusi }
     biaya i : integer
     biaya j : integer
     biaya k : integer
     biaya l : integer

     algoritma:
     S<-- {} { himpunan solusi }
     while (belum ada transport di S)
       do
```

```
cek transportasi yang melewati
     simpul awal
cek transportasi yang melewati
     simpul akhir
cek simpul yang dilewati transport
if ( simpul awal dan akhir ada
pada transport dengan biaya
terkecil) then
     himpunan solusi <-- transport
else if (hanya simpul awal/simpul
akhir terdapat pada transport
dengan biaya terkecil) then
     if (simpul awal yg terdapat
     pd transport dengan
     biaya terkecil ) then
            cari transport biaya
            terkecil
            setelahnya yang
            mengarah ke simpul
            tujuan
            masukkan transport ke
            solusi
            if (transport
            setelahnya     mencakup
            kedua simpul) then
                  himpunan solusi
                  <--    kombinasi
                  transport
            else
                  greedycost
                  himpunan solusi
                  <--    kombinasi
                  transport dengan
                  cost terkecil

     else
            cari transport biaya
            terkecil     setelahnya
            yang
            mengarah ke simpul awal
            masukkan transport ke
            solusi
            if (transport
            setelahnya     mencakup
            kedua simpul) then
                  himpunan solusi
                  <--    kombinasi
                  transport
            else
                  greedycost
                  himpunan solusi
                  <--    kombinasi
                  transport dengan
                  cost terkecil
```

Contoh kasus 1:

Rute A ke F

Langkah-langkah pemecahan masalah pemilihan angkutan untuk rute dari simpul A ke simpul F dengan algoritma *greedy* biaya adalah sebagai berikut:

- Cek transportasi yang melewati simpul A (transportasi I dan J)
- Cek transportasi yang melewati simpul F (transportasi J dan transportasi K)
- Cek biaya masing-masing transportasi :
- Transport I : 4000, transport J : 4000, transport K : 3000
- Biaya yang paling kecil adalah K = 3000 namun karena K tidak melewati simpul A maka yang dipilih sebagai solusi adalah transport J dengan biaya 4000 (karena transport J + K > 4000) (biaya optimal: 4000) dan panjang lintasan 12 (Panjang lintasan optimal : 10)

Contoh kasus 2:

Rute E ke G

Langkah-langkah pemecahan masalah pemilihan angkutan untuk rute dari simpul E ke simpul G dengan algoritma *greedy* biaya adalah sebagai berikut:

- Cek transportasi yang melewati simpul E (transport J dan K).
- Cek transportasi yang melewati simpul G (transport L)
- Cek simpul yang dilewati ketiga transport.
  Transport J (A, C, D, E, F).
  Transport K (D,E,F).
  Transport L (C, G)
- Cek biaya masing-masing transport : transport J : 4000, transport K : 3000, transport L : 2000
- Karena simpul akhir hanya dilewati satu transport maka transport L dimasukkan ke dalam himpunan solusi
- Selanjutnya yang dapat diambil adalah simpul J dan/atau K
- Cek apakah simpul yang ada di transport L ada di kedua transport sebelumnya (ada: transport J). Masukkan J ke himpunan solusi
- Cek apakah simpul awal terdapat pada kedua transport (ada, simpul E pada transport J).
- Himpunan solusi: transport J dan transport L dengan biaya sebanyak (4000 + 2000) = 6000, (biaya optimal: 6000) panjang lintasan: 15 (panjang lintasan optimal : 15).

Contoh kasus 3:

Rute D ke C

Langkah-langkah pemecahan masalah pemilihan angkutan untuk rute dari simpul D ke simpul C dengan algoritma *greedy* biaya adalah sebagai berikut:

- Cek transportasi yang melewati simpul D (transport J dan transport K)
- Cek transportasi yang melewati simpul C (transport I, J, L).
- Cek simpul yang dilewati ketiga transport.
  Transport I (A, B,C)
  Transport J (A, C, D, E, F).
  Transport K (D,E,F).
  Transport L (C, G)
- Cek biaya masing-masing transport : transport I : 4000, transport J : 4000, transport K : 3000, transport L : 2000
- Biaya transport terkecil: transport L = 2000
- Cek apakah simpul di L (C, G) ada yang

dicakup pada transport J atau K. (C ada pada transport J).

- Masukkan simpul J ke himpunan solusi
- Himpunan solusi : transport J dengan biaya 4000 (biaya optimal: 4000) Panjang lintasan : 14 (panjang lintasan optimal: 12).

IV. PENDEKATAN DENGAN ALGORITMA BRANCH AND BOUND

Penggunaan algoritma *branch and bound* pada penentuan pemilihan rute transportasi yang akan dibahas pada makalah ini adalah untuk masalah *shortest path*.

Dengan prinsip *branch and bound*, cost yang ditentukan untuk algoritma ini adalah jauh dari sebuah simpul ke simpul tujuan serta banyak simpul terkecil yang harus dihidupkan untuk menuju simpul tujuan.

Alur algoritma *branch and bound* yang digunakan adalah sebagai berikut:

- Tentukan simpul awal serta simpul tujuan
- Hidupkan simpul yang bertetangga dengan simpul awal
- Cek cost yang diperlukan dari masing masing simpul yang dihidupkan
- Apabila simpul yang dihidupkan sudah pernah dihidupkan sebelumnya maka simpul tersebut di-*bound*.

Pada contoh seperti pada gambar G3.1 alur algoritma dijelaskan seperti berikut untuk pemilihan rute dari A ke F:

- simpul A dihidupkan, masukkan ke dalam himpunan solusi
- menghidupkan B, C, dan D
- cost B : 28, cost C : 16, cost D : 7
- masukkan simpul D ke himpunan solusi
- menghidupkan simpul A, B, F, G
- masukkan simpul F ke himpunan solusi
- simpul tujuan sudah ada di himpunan solusi, selesai

Rute yang terbentuk: A – D – F (panjang lintasan: 10)

V. KESIMPULAN

Dari hasil analisis yang dibuat pada makalah ini maka dapat disimpulkan bahwa pemilihan rute *shortest path* dengan menggunakan *greedy* tidak efisien dikarenakan prinsip *greedy* hanya mengambil keputusan optimum lokal dan mengabaikan pilihan rute-rute lain yang mungkin menghasilkan rute yang lebih pendek.

Sedangkan untuk algoritma *greedy* biaya, kesimpulan yang didapatkan adalah untuk meminimalisasi biaya, algoritma sudah cenderung mengarah ke biaya optimal, namun untuk algoritma *greedy* biaya ini tidak memperhatikan unsur panjang lintasan sehingga panjang lintasan yang didapat mungkin tidak optimal.

Pada algoritma *branch and bound* dengan cost yang terdefinisi hasilnya sudah terhitung cukup optimal. Sehingga untuk permasalahan rute angkot terpendek penyelesaiannya lebih efisien menggunakan algoritma *Branch-and-bound* daripada *greedy*.

REFERENSI

Munir, Rinaldi. (2007). Diktat Kuliah Strategi Algoritma, Departemen Teknik Informatika, Institut Teknologi Bandung.
url: http://www.cs.berkeley.edu/~vazirani/algorithms/chap5.pdf
Tanggal akses: 18 Mei 2014 pukul 13:12

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 2014

Arieza Nadya (135012017)