Analisis Beberapa Algoritma dalam Menyelesaikan Pencarian Jalan Terpendek

Hugo Toni Seputro

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika
Jl. Ganesha 10 Bandung Jawa Barat Indonesia
If16053@students.if.itb.ac.id

ABSTRAK

Kita semua ingin dalam setiap waktu ingin mencapai tempat tujuan kita dengan melalui jalan yang sependek mungkin, untuk menghemat waktu. Untuk mendapati jalan terpendek ini kita biasanya menggunakan insting kita dalam menentukan jalan mana yang lebih pendek. Namun, dalam usaha untuk mencari jalan terpendek ini sering kali insting kita salah, untuk itulah ita pergunakan algoritma.
Setelah menyelesaikan tugas kedua untuk mata kuliah Strategi Algoritmik, yang tugasnya merupakan penyelsaian masalah mencari lintasan terpendek dari start sampai finish di dalam labirin, saya mendapatkan bahwa masalah mencari lintasan terpendek ini dapat diselesaikan oleh banyak algoritma yang berbeda yang akan memberikan keefesienan yang berbeda pula. Makalah ini membahas tentang beberapa algoritma yang dapat menyelesaikan masalah lintasan terpendek tersebut. Algoritma yang akan dibahas adalah algoritma *greedy*, algoritma *Branch & Bound, dan* algoritma *BackTracking*,. Tujuan dari dibuatnya makalah ini adalah agar pembaca dapat saling melihat perbedaan antara metode-metode algoritma yang ada dalam usahanya untuk memecahkan masalah pencarian lintasan terpendek ini.

Kata kunci: Lintasan Terpendek, *Greedy*, *Branch&Bound*, *BackTracking*.

1. PENDAHULUAN

Definisi dari lintasan terpendek disini bervariasi, dalam makalah ini kita akan membahas lintasan terpendek dalam masalah Traveling Sales Person (TSP), lintasan terpendek dalam suatu labirin, dan lintasan terpendek antara dua simpul dalam graph.

2. Algoritma Greedy

2.1 Definisi Algoritma Greedy

Algoritma Greedy membentuk solusi langkah per langkah. Pada setiap langkah tersebut akan dipilih keputusan yang paling optimal. Keputusan tersebut tidak perlu memperhatikan keputusan selanjutnya yang akan diambil, dan keputusan tersebut tidak dapat diubah lagi pada langkah – langkah selanjutnya. Prinsip utama algoritma greedy adalah prinsip *"take what you can get now!"*. Maksud dari prinsip tersebut adalah sebagai berikut; pada setiap langkah dalam algoritma greedy kita ambil keputusan yang paling optimal untuk langkah tersebut tanpa memperhatikan konsekuensi pada langkah – langkah selanjutnya, lalu kita namakan solusi tersebut dengan optimum lokal**.** Dengan cara pengambilan nilai optimum lokal pada setiap langkah diharapkan akan tercapainya optimum global**,** yaitu tercapainya solusi optimum yang melibatkan keseluruhan langkah dari awal sampai akhir.

2.2. Skema Umum Algoritma Greedy

Elemen – elemen yang digunakan dalam penerapan algoritma greedy antara lain :

1. Himpunan Kandidat, C.
   Himpunan yang berisi elemen pembentuk solusi.
2. Himpunan Solusi, S.
   Himpunan yang terpilih sebagai solusi persoalan.
3. Fungsi Seleksi, SELEKSI.
   Fungsi yang memilih kandidat yang paling mungkin untuk mencapai solusi optimal.
4. Fungsi Kelayakan, LAYAK.
   Fungsi yang memeriksa apakah suatu kandidat yang dipilih dapat memberikan solusi yang layak. Maksudnya yaitu apakah kandidat tersebut bersama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala yang ada.
5. Fungsi SOLUSI
   Fungsi yang mereturn *boolean.* True jika himpunan solusi yang sudah tebentuk merupakan solusi yang

lengkap; False jika himpunan solusi belum lengkap.

6. Fungsi Objektif
   Fungsi yang mengoptimalkan solusi.

Skema umum dari algoritma greedy dapat kita tuliskan sebagai berikut :

1. Inisialisasi S dengan kosong.
2. Pilih sebuah kandidat dari C (dengan SELEKSI).
3. Kurangi C dengan kandidat yang telah terpilih di atas.
4. Periksa apakah kandidat yang dipilih tersebut bersama – sama dengan himpunan solusi membentuk solusi yang layak (dengan LAYAK). Jika ya, masukkan kandidat ke himpunan solusi; jika tidak buang kandidat tersebut dan tidak perlu ditelaah lagi.
5. Periksa apakah himpunan solusi yang sudah terbentuk telah memberikan solusi yang lengkap (dengan SOLUSI). Jika ya, berhenti; jika tidak, ulangi dari langkah 2.

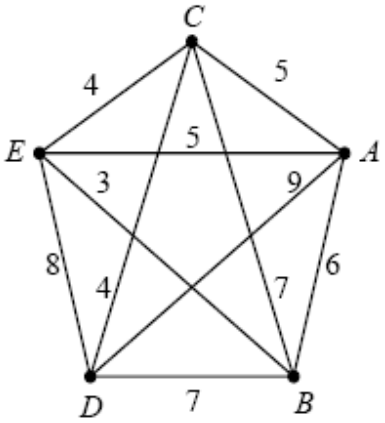
2.3. Penyelesaian TSP dengan Algoritma Greedy

Kita akan menerapkan algoritma *greedy* pada persoalan *Traveling Salesman Problem* (*TSP*). Kita juga dapat melihat kompleksitas dari algoritma tersebut.
Disini permasalahan TSP yang ingin dianalisa adalah sebagai berikut:
Seorang sales harus mengunjungi n kota, melewati setiap kota sebanyak 1 kali, dimulai dari salah satu kota yang didefinisikan sebagai tempat asalnya dan harus kembali ke kota tersebut. Biaya transportasi antar kota tersebut telah diberikan. Program ini merepresentasikan kunjungan sales tersebut ke kota-kota yang harus dilewati dengan biaya minimum. Penyelesaian *TSP* adalah sebagai berikut: Misal kota-kota tersebut direpresentasikan dengan angka 1 hingga n, kota 1 sebagai kota awal dimana salesman memulai perjalanannya. Asumsikan fungsi c(i,j) adalah biaya kunjungan dari kota i ke j. Ada kemungkinan biaya c(i,j) tidak sama dengan biaya c(j,i). Kemungkinan solusi yang pasti adalah (n-1)!. Seseorang mungkin dapat menurunkannya secara sistematis, menemukan biaya untuk setiap masalah dan akhirnya mempunyai solusi dengan biaya terendah. Ini membutuhkan setidaknya (n-1)! langkah.

Penyelesaian masalah TSP dengan algoritma *greedy* ialah, jika salesman ada di kota i, dia akan memilih kota selanjutnya, j dimana c(i,j) merupakan biaya terkecil dari semua biaya c(i,k), dimana k merupakan kota-kota yang belum pernah dikunjungi.

- Mulai dari sembarang kota.
- Evaluasi semua biaya tetangga.
- Ambil tetangga dengan biaya terkecil dan diulang pada langkah ke dua hingga kota telah terlewati semua.

Contoh :



Gambar1. TSP dengan n = 5

Langkah 1: Dari *E* pergi ke *B* (Total jarak = 3)
Langkah 2: Dari *B* pergi ke *A* (Total jarak = 3 + 6 = 9)
Langkah 3: Dari *A* pergi ke *C* (Total jarak = 9 + 5 = 14)
Langkah 4: Dari *C* pergi ke *D* (Total jarak = 14 + 4 = 18)
Langkah 5: Dari *D* kembali lagi ke *E* (Total jarak = 18 + 8 = 26)
Perjalanan (sirkuit Hamilton) yang dihasilkan:
$E \rightarrow B \rightarrow A \rightarrow C \rightarrow D \rightarrow E$
Panjang = 3 + 6 + 5 + 4 + 8 = 26 (tidak optimal)
Optimal :
$E \rightarrow B \rightarrow D \rightarrow C \rightarrow A \rightarrow E$
panjang = 3 + 7 + 4 + 5 + 5 = 24

Dalam tiap iterasi, dicari kota tetangga yang merupakan suksesornya, maka kompleksitasnya : $O(n^2)$.

Algoritma *greedy*, biasanya linier ke kuadrat dan memerlukan memori ekstra sedikit/kecil. Namun pada umumnya algoritma ini tidak selalu benar. Algoritma *greedy*

tidak selalu memberikan solusi optimum. Tetapi ketika algoritma *greedy* bekerja, lebih mudah untuk diterapkan dan cukup cepat untuk dilaksanakan.

3. Algoritma Branch&Bound

3.1 Definisi Algoritma Branch&Bound

Algoritma Branch and Bound (B&B) adalah metode pemecahan persoalan optimasi . Algoritma B&B merupakan penerapan dan perbaikan dari metode BFS (Breadth First Search) dimana simpul yang akan dibangkitkan adalah simpul hidup yang memiliki cost paling kecil . Secara harafiah Branch artinya cabang pohon dan Bound artinya mengikat.

Algoritma Branch and Bound (B&B) merupakan metode pencarian di dalam ruang solusi secara sistematis . Ruang solusi diorganisasikan ke dalam pohon ruang status . Pembentukan pohon ruang status pada B&B adalah dengan skema BFS . Simpul – simpul pada B&B diberi sebuah nilai cost . Kemudian simpul – simpul berikutnya dibangkitkan berdasarkan simpul hidup yang memiliki cost terkecil . Nilai okngkos (c(i)) pada setiap simpul I menyatakan taksiran ongkos termurah lintasan dari simpul I ke simpul solusi.

3.2. Skema Umum Algoritma Branch&Bound

Langkah – langkah pemecahan masalah dengan algortima Branch and Bound :

1. Membangkitkan simpul-simpul kemungkinan atas langkah-langkah yang dapat diambil dari kondisi / simpul awal .
2. Terapkan suatu fungsi untuk menghitung ongkos / cost setiap simpul untuk mencapai solusi .
3. Simpul dengan ongkos terkecil akan dibangkitkan simpul-simpul anaknya .
4. Lalu lakukan perbandingan simpul-simpul hidup mana yang memiliki cost terkecil .
5. Ulang kembali ke langkah ketiga sampai simpul solusi ditemukan .

Dengan hanya dibangkitkannya simpul-simpul dengan cost terkecil tersebut maka diharapkan bahwa ketika mencapai solusi langkah-langkah yang diambil telah optimal .

3.3. Penyelesaian Lintasan Terpendek dalam Labirin dengan Algoritma Branch&Bound

Contoh permasalahan ini diambil dari tugas2 mata kuliah Strategi Algoritmik kelompok saya, diberikan permasalahan bahwa : didalam sebuah labirin berukuran 8x8 kotak, yang terdapat sejumlah tembok pada posisi tertentu yang ditentukan user, sebuah kotak start dan sebuah kotak finish. Permasalahan terletak pada, pencarian jalan terpendek antara kotak start dengan kotak finish, dimana jalan tersebut tidak boleh melalui tembok.

Langkah pemecahan umum yang akan dipakai adalah :

1. Robot akan berada pada kotak biru yang merupakan posisi awal.
2. Bangkitkan simpul anak dari akar.
3. Simpul anak yang dibangkitkan akan menjadi simpul hidup dan bila suatu simpul , simpul anaknya sudah dibangkitkan maka simpul tersebut bukan lagi simpul hidup.
4. Pilih simpul anak yang memiliki ongkos yang paling kecil dari simpul hidup lainnya dan periksa apakah simpul ini sudah merupaka simpul solusi , bila belum bangkitkan simpul ini. Bila sudah merupakan simpul akhir maka pencarian dihentikan.
5. Bila simpul-simpul anak memiliki ongkos yang sama maka simpul yang memiliki jarak x terdekatlah yang akan dibangkitkan.
6. Ulang kembali tahap 3.
7. Setiap jalan yang dilalui robot akan berubah warna menjadi merah.
8. Solusi ditemukan bila robot sudah mencapai posisi akhir.

Contoh masalah :

Gambar2. Labirin dengan Branch&Bound

Langkah-langkah pemecahan masalah :

1. Posisi awal ada pada titik 2,1 , lalu simpul anaknya dibangkitkan.
2. Simpul anaknya adalah 3,1 dan 1,1 . Ongkos dari 3,1 adalah 6 dan ongkos dari 1,1 adalah 8 maka simpul 3,1 yang dibangkitkan simpul anaknya karena 3,1 bukan merupakan simpul solusi.
3. Simpul anak 3,1 adalah 4,1 dengan ongkos 8 karena 4,1 bukan solusi dan merupakan simpul dengan ongkos terkecil saat ini maka simpul anak 4,1 dibangkitkan.
4. Simpul anak 4,1 adalah 5,1 dengan ongkos sebesar 10, sedangkan simpul hidup lainnya 1,1 memiliki ongkos 8 , maka simpul anak 1,1 dibangkitkan.
5. Simpul anak 1,1 adalah 1,2 dengan ongkos 8 , karena 1,2 bukan solusi maka simpul anaknya dibangkitkan.
6. Simpul anak 1,2 adalah 1,3 dengan ongkos 8 , karena 1,3 bukan solusi maka simpul anaknya dibangkitkan.
7. Simpul anak 1,3 adalah 2,3 dengan ongkos 8 , karena 2,3 bukan solusi maka simpul anaknya dibangkitkan.
8. Simpul anak 2,3 adalah 3,3 dengan ongkos 8 , karena 3,3 bukan solusi maka simpul anaknya dibangkitkan.
9. Simpul anak 3,3 adalah 4,3 dengan ongkos 10 dan 3,4 dengan ongkos 8 , karena 3,4 bukan solusi maka simpul anaknya dibangkitkan.
10. Simpul anak 3,4 adalah 4,4 dengan ongkos 10 dan 3,5 dengan ongkos 8 , karena 3,5 bukan solusi maka simpul anaknya dibangkitkan.
11. Simpul anak 3,5 adalah 4,5 dengan ongkos 10 dan 3,6 dengan ongkos 8 , karena 3,6 merupakan solusi maka pencarian dihentikan.
12. Jadi jalur yang diambil dari posisi awal sampai posisi akhir adalah (1,1),(1,2),(1,3),(2,3),(3,3),(3,4),(3,5) ,dan (3,6).

Kesimpulan yang didapat dari berbagai pengujian adalah algoritma Branch and Bound akan menghasilkan hasil yang selalu optimal. Hal ini disebabkan pada algoritma langkah-langkah yang diambil hanya langkah-langkah yang memiliki ongkos yang optimal, sehingga setelah semua algoritma diambil hasil yang didapat pasti optimal. Dari segi kompleksitas maka algoritma Branch and Bound lebih baik daripada algoritma brute force, walaupun keduanya menghasilkan solusi yang optimal dan algoritma membutuhkan berbagai struktur atau kelas untuk mengimplementasikannya.

3. Algoritma BackTracking

3.1 Definisi Algoritma BackTracking

Algoritma BackTracking atau runut-balik merupakan salah satu metode dalam pemangkasan pohon ruang status. Algortima ini merupakan kembangan dari metode DFS (*Deep First Search*) yang kemudian memiliki fungsi batas (*constraints*) agar pencarian yang dilakukan lebih mengarah kepada solusi. Fungsi batas merupakan faktor yang akan menyebabkan program untuk melakukan runutbalik.

3.2. Skema Umum Algoritma BackTracking

Dalam penerapan algoritma runut balik (*backtracking*) dibutuhkan properti-properti berikut :

1. Solusi Persoalan
   Solusi dari persoalan dinyatakan dalam sebuah vektor $Y = (y1 , y2, ..., yn)$, $yi \hat{I} Si$. Si tidak harus berbeda. Si merupakan himpunan nilai yi yang merupakan kemungkinan dari solusi. Misalkan $Si = \{a, b\}$ maka $yi = a$ atau b.
2. Fungsi Pembangkit nilai yk.
   Fungsi untuk membangkitkan nilai untuk yk yang merupakan komponen dari vektor solusi.
   MAKALAH IF2251 STRATEGI ALGORITMIK TAHUN 2007
3. Fungsi Pembatas
   Fungsi yang menentukan apakah ($y1 , y2, ..., yk$) mengarah pada vektor solusi atau tidak. Jika ya, maka pembangkitan untuk nilai $yk+1$, tetapi jika tidak maka ($y1 , y2, ..., yk$) dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi[3].

Langkah-langkah pencarian solusi dalam *BackTracking* adalah sebagai berikut :

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti metode pencarian mendalam (DFS). Simpul-simpul yang sudah dilahirkan dinamakan simpul hidup (*live node)*.Simpul hidup yang sedang diperluas dinamakan simpul-E (*expand node)*.
2. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang. Jika lintasan yang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut dibunuh sehingga menjadi simpul mati (*dead node*). Fungsi yang digunakan untuk membunuh

simpul-E adalah dengan menerapkan fungsi pembatas (*bounding function*).

3. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan runut-balik ke simpul hidup terdekat (simpul bapak). Selanjutnya, simpul ini menjadi simpul-E yang baru. Lintasan baru dibangun kembali sampai lintasan tersebut membentuk solusi.
4. Pencarian dihentikan apabila tidak ada lagi simpul hidup untuk runut-balik.

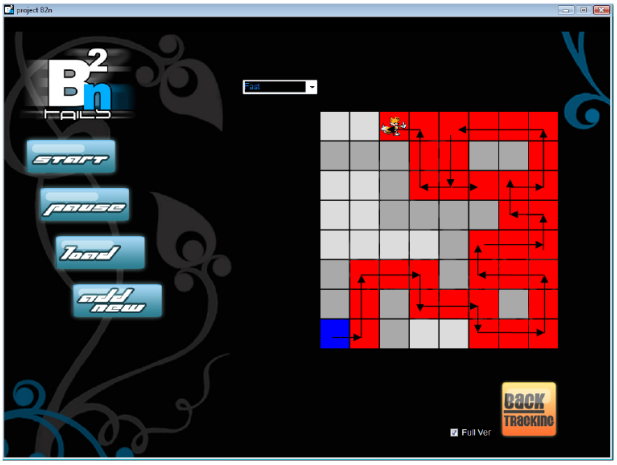
3.3. Penyelesaian Lintasan Terpendek dalam Labirin dengan Algoritma BackTracking

Contoh yang sama dengan Algoritma Branch&Bound kembali diambil di Algoritma BackTracking. Permasalahannya sama dengan sebelumnya : didalam sebuah labirin berukuran 8x8 kotak, yang terdapat sejumlah tembok pada posisi tertentu yang ditentukan user, sebuah kotak start dan sebuah kotak finish. Permasalahan terletak pada, pencarian jalan terpendek antara kotak start dengan kotak finish, dimana jalan tersebut tidak boleh melalui tembok.

Langkah-langkah umum BackTracking :

1. Dari simpul akar , bangkitkan simpul anak-anaknya.
2. Simpul anak yang mengarah kekanan diambil.
3. Bangkitkan simpul anak yang mengarah ke kanan sampai simpul anak tersebut tidak bias lagi bergerak ke kanan.
4. Bila simpul anak tidak bias lagi bergerak ke kanan maka bangkitkan simpul anak yang bergerak ke bawah.
5. Bila Simpul anak tidak bias bergerak ke kanan ataupun ke bawah maka bangkitkan simpul anak yang bergerak ke kiri.
6. Bila Simpul anak tidak bias bergerak ke kanan, bawah ataupun kiri, maka bangkitkan simpul anak yang bergerak ke atas.
7. Bila simpul anak mencapai tempat dimana dia tidak bias bergerak kemana-mana, dengan asumsi bahwa dia tidak boleh bergerak kea rah yang sama dengan tempat dia datang, maka lakukan backtracking sampai percabangan terakhir dilakukan.
8. Lakukan terus langkah 2 sampai 7 sampai robot mencapai finish.
9. Simpul yang telah dilalui 2 kali (yang sudah pernah di lakukan backtracking) diberi flag, agar tidak dilewati untuk ketiga kalinya.
10. Pencarian berhenti setelah mencapai simpul solusi.

Contoh permasalahan :



Gambar3. Labirin dengan BackTracking

Langkah-langkah pemecahan masalah :

1. Pada contoh kasus pada gambar di atas dapat diketahui bahwa posisi awal pada titik 1,8 dan posisi akhir pada 3,1.
2. Simpul yang mengarah kekanan dibangitkan terus sampai robot tidak bisa bergerak kekanan lagi (Sampai pada posisi 2,8).
3. Apabila robot tidak bisa bergerak kebawah dan kekiri, , maka ia akan membangkitkan simpul yang mengarah keatas, sambil mencek apakah ia bisa bergerak kekanan, atau kekiri, diulangi terus menerus sampai robot tidak bias bergerak ke atas lagi (Sampai pada posisi 2,6).
4. Simpul yang mengarah kekanan dibangitkan terus sampai robot tidak bisa bergerak kekanan lagi (Sampai pada posisi 4,6).
   IF2251 Strategi Algoritmik 36
5. Bangkitkan simpul yang mengarah kebawah sembari terus menerus mencek apakah robot dapat mengarah kekanan atau tidak, bila tidak bisa, maka robot akan bergerak ke bawah, langkah ini diulangi sampai robot tidak bisa bergerak kebawah lagi (Sampai pada posisi 4,7).
6. Karena robot dapat melewati arah kekanan, maka simpul kekanan dibangkitkan sampai robot

tidak dapat bergerak kekanan lagi (Sampai pada posisi 6,7).

7. Bangkitkan simpul yang mengarah kebawah sembari terus menerus mencek apakah robot dapat mengarah kekanan atau tidak, bila tidak bisa, maka robot akan bergerak ke bawah, langkah ini diulangi sampai robot tidak bisa bergerak kebawah lagi (Sampai pada posisi 6,8).
8. Karena robot dapat melewati arah kekanan, maka simpul kekanan dibangkitkan sampai robot tidak dapat bergerak kekanan lagi (Sampai pada posisi 8,8).
9. Apabila robot tidak bisa bergerak kebawah dan kekiri, maka ia akan membangkitkan simpul yang mengarah keatas, sambil mencek apakah ia bisa bergerak kekanan, atau kekiri, diulangi terus menerus sampai robot tidak bias bergerak ke atas lagi (Sampai pada posisi 8,6).
10. Bangkitkan simpul yang mengarah kekiri sambil terus mencek apakah ia dapat bergerak kebawah, lakukan terus menerus (Sampai pada posisi 6,6).
11. Karena robot dapat bergerak ke atas, maa simpul yang mengarah keatas dibangkitkan, (Posisi 6,5). Dan robot akan bergerak keatas kemudian kekanan sampai pada posisi 8,5.
12. Dari posisi 8,5 robot akan ,mengarah keatas sampai posisi 8,4. Kemudian kekiri sampai posisi 7,4. Dan keatas lagi sampai posisi 7,3.
13. Dari posisi 7,3 robot bergerak kekanan, ke posisi 8,3. Karena simpul pada posisi 8,4 sudah pernah dibangkitkan, maka robot bergerak keatas sampai posisi 8,1.
14. Robot akan Membangkitkan simpul yang mengarah kekiri sambil terus mencek apakah ia dapat bergerak kebawah, lakukan terus menerus (Sampai pada posisi 5,1).
15. Bangkitkan simpul yang mengarah kebawah sembari terus menerus mencek apakah robot dapat mengarah kekanan atau tidak, bila tidak bisa, maka robot akan bergerak ke bawah, langkah ini diulangi sampai robot tidak bisa bergerak kebawah lagi (Sampai pada posisi 5,3).
16. Bangkitkan simpul yang mengarah kekanan, ketika robot tidak dapat bergerak lagi (karena posisi kanan dari 6,3 sudah pernah dilalui) maka dilakukan backtrack sampai percabangan terdekat (Pada posisi 5,3).
17. Robot akan bergerak ke kiri, karena arah atas sudah pernah dilalui (Sampai posisi 4,3).
18. Simpul yang mengarah keatas dibangkitkan karena robot tidak dapat bergerak kekiri maupun kebawah, sembari terus melakukan pengecekan apakah robot dapat bergerak kekanan atau kekiri (Sampai posisi 5,1).
19. Robot bergerak kekiri sampai finish.

IV. KESIMPULAN

Kesimpulan yang didapat dari berbagai pengujian adalah algoritma Branch and Bound akan menghasilkan hasil yang selalu optimal. Hal ini disebabkan pada algoritma langkah-langkah yang diambil hanya langkah-langkah yang memiliki ongkos yang optimal , sehingga setelah semua algoritma diambil hasil yang didapat pasti optimal. Sedangkan untuk algoritma runut balik solusi akan selalu ditemukan (bila ada solusi) walaupun solusi itu belum tentu optimal , kecuali pada algoritma runut balik diberi nilai ongkos pada setiap langkahnya , walaupun hal ini membuat algoritma menjadi sangat mirip dengan algoritma Branch and Bound. Dan untuk algoritma greedy, walaupun algoritma ini tidak selalu memberikan hasil yang optimum, namun algoritma greedy lebih mudah, sederhana, dan cepat memperoleh hasil ketika digunakan, sehingga menjadi salah satu algoritma paling favorit.

Dari segi kompleksitas maka algoritma Branch and Bound lebih baik daripada algoritma brute force, walaupun keduanya menghasilkan solusi yang optimal dan algoritma membutuhkan berbagai struktur atau kelas untuk mengimplementasikannya.

REFERENSI

[1] Munir, Rinaldi. 2005. Strategi Algoritmik. Teknik Informatika ITB : Bandung

[2] http://en.wikipedia.org/wiki/Greedy_algorithm