PENGARUH PEMILIHAN PEMBANGKITAN ANAK DENGAN COST SAMA PADA KASUS PENCARIAN LINTASAN TERPENDEK MENGGUNAKAN ALGORITMA BRANCH AND BOUND

David Samuel

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
e-mail: david_samuel@students.itb.ac.id

ABSTRAK

Algoritma *Branch and Bound* adalah salah satu algoritma yang dapat memberikan solusi optimal dalam setiap penggunaannya. Salah satu kegunaan alogritma *Branch and Bound* adalah untuk menentukan lintasan terpendek yang dapat ditempuh dari lantai *start* menuju lantai *finish* dalam sebuah labirin (*maze*). Dalam kasus pencarian lintasan terpendek, secara sistematis algoritma *Branch and Bound* akan memilih setiap langkah yang didasarkan pada kalkulasi *cost* setiap lantai dengan menggunakan fungsi heuristik.

Secara garis besar, algoritma *Branch and Bound* akan menggunakan bantuan sebuah pohon BFS, dan memodifikasinya dengan memperluas simpul yang memiliki *cost* terkecil terlebih dahulu, dengan tujuan untuk mendapatkan solusi secara lebih cepat dan optimal dibandingkan dengan memperluas seluruh simpul yang berada pada tingkat yang sama, yang diterapkan dalam perluasan pohon *BFS*.

Akan tetapi, tidak ada aturan yang pasti mengenai urutan pembangkitan simpul saat ada dua buah simpul atau lebih yang memiliki cost sama dan merupakan cost terkecil dibandingkan simpul-simpul lainnya. Makalah ini akan membahas pengaruh pemilihan simpul yang diperluas terlebih dahulu, apabila terjadi hal seperti demikian, dengan menggunakan contoh kasus pencarian lintasan terpendek.

Kata kunci: BFS, *Branch and Bound*, *cost*, heuristik,*maze*

1. PENDAHULUAN

Algoritma *Branch and Bound* menggunakan bantuan pohon BFS dalam mencari solusi optimalnya. Algoritma murni BFS akan menggunakan antrian untuk menyimpan simpul yang dibangkitkan. Simpul-simpul yang baru dibangkitkan akan diletakkan pada antrian dengan prinsip FIFO (*First In First Out*). Sedangkan simpul-E, yaitu simpul hidup yang akan diekspansi berikutnya adalah simpul yang pertama masuk ke dalam antrian.

Namun, algoritma murni BFS ini kalah cepat dibandingkan dengan apabila kita mengekspansi pohon BFS dengan algoritma *Branch and Bound*. Kunci utama algoritma *Branch and Bound* adalah pemilihan simpul yang akan diekspansi secara selektif berdasarkan *cost* untuk mencapai simpul tersebut. Simpul dengan *cost* terkecil, dan mengarah ke simpul solusi, akan dibangkitkan terlebih dahulu, sekaligus mematikan simpul-simpul yang tidak mengarah ke simpul solusi.

Setiap simpul harus diberi nilai terlebih dahulu. Pemberian nilai didasarkan pada fungsi heuristik, yaitu sebuah fungsi yang menyatakan perkiraan nilai. Tidak ada sebuah pernyataan matematika yang dapat merepresentasikan fungsi heusristik secara pasti, namun fungsi heuristik diturunkan berdasarkan pengamatan terhadap permasalahan yang spesifik.

Ada sebuah algoritma yang mirip dengan algoritma *Branch and Bound* , algoritma tersebut dikemukakan oleh Hart, Nilsson, dan Raphael (1968), yang dikenal sebagai algoritma A*. Pada perkembangannya sekarang telah memiliki banyak varian, seperti AlphaA*, dan sebagainya, namun pada makalah ini, akan digunakan istilah Algoritma *Branch and Bound,* dikarenakan aplikasi pemakaiannya yang lebih luas, sedangkan penggunaan kasus pencarian lintasan terpendek hanya untuk menunjukkan salah satu kasus urutan pembangkitan pada pohon BFS jika ada dua simpul yang bernilai sama.

2. PENCARIAN JALUR TERPENDEK

2.1 Batasan Masalah

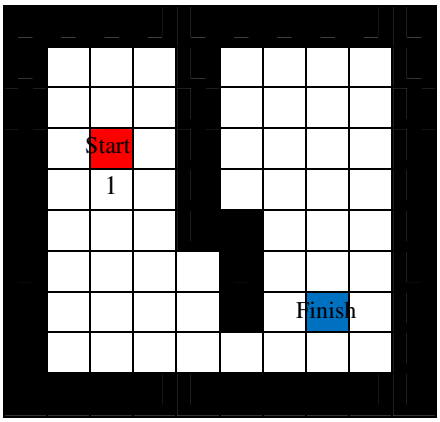
Untuk merepresentasikan penggunaan algoritma *Branch and Bound* serta cara ekspansi simpul pada pohon BFS, akan digunakan contoh permasalahan mencari lintasan terpendek dalam sebuah labirin.

Diberikan sebuah labirin yang terdiri dari lantai-lantai kotak, dengan ukuran labirin 8x8. Pada labirin terdapat sebuah lantai yang didefinisikan sebagai *start*, dan sebuah lantai yang didefinisikan sebagai *finish.* Sebuah robot pencari jejak ditempatkan di lantai *start* dan ditugaskan untuk mencari lintasan terpendek, melewati tiap lantai dan menghindari halangan tembok pada labirin, sehingga robot dapat mencapai finish.

Batasan yang diberikan pada permasalahan ini adalah robot hanya dapat bergerak vertikal dan horizontal, serta tidak dapat menembus tembok.

Adapun struktur data lantai yang digunakan adalah sebagai berikut:

```
Type Lantai
  c, f, g: integer{Nilai fungsi heuristik}
  Parent: Lantai {Penunjuk ke lantai induk}
  Start,Finish: Boolean
  Wall,Path : Boolean
```



Gambar 2.1 Pencarian Jalur Terpendek Untuk Labirin 8x8

2.2 Fungsi Heuristik Manhattan

Setiap lantai yang terdapat dalam labirin akan diberikan nilai yang merepresentasikan nilai simpul pada pohon pemebentukannya. Oleh karena itu akan digunakan fungsi heuristik untuk memperkirakan nilai setiap lantai yang akan dilewati oleh

$$\hat{c}(i) = \hat{f}(i) + \hat{g}(i) \qquad (1)$$

Di mana $\hat{c}(i)$ = ongkos untuk simpul i

$\hat{f}(i)$ = ongkos untuk mencapai lantai i dari *start*

$\hat{g}(i)$ = ongkos untuk mencapai finish dari lantai i

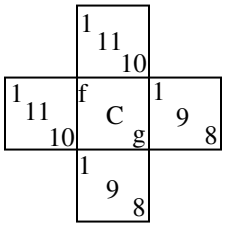
Pada permasalahan pencarian jalur terpendek ini, akan digunakan fungsi heuristik Manhattan untuk mengkalkulasi *cost* setiap lantai. Fungsi Manhattan didefiniskan sebagai berikut:

$$\hat{f}(i) = \hat{f}(parent) + 1 \qquad (2)$$

$\hat{f}(i)$ adalah ongkos yang didasarkan pada lantai sebelumnya. Ini adalah perkiraan pencapaian lantai i, jika dihitung dari lantai start.

$$g(i) = abs(finish.x - i.x) + abs(finish.y - i.y) \qquad (3)$$

$g(i)$ adalah ongkos yang dikalkulasi berdasarkan nilai absolut posisi lantai tersebut terhadap posisi lantai finish, perlu diketahui bahwa fungsi $g(i)$ akan menganggap tembok seperti layaknya lantai biasa, sehingga nilai ongkos lantai adalah nilai absolut posisi lantai terhadap lantai finish. Sehingga, ilustrasi yang tepat untuk lantai nomor 1 pada labirin gambar 2.1 adalah:



Gambar 2.2 Ilustrasi pemberian *cost* lantai untuk lantai nomor 1

Proses pemberian nilai ini akan terus diberikan seiring dengn pemilihan jalur yang dilintasi oleh robot.

2.3 Proses Pembangkitan Simpul

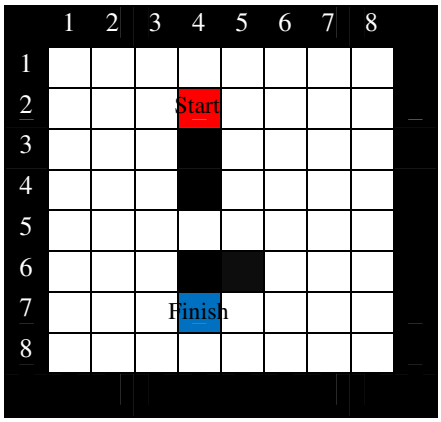
Proses pembangkitan simpul menggunakan antrian Q, di mana antrian yang digunakan adalah antrian Q yang terurut menaik, sehingga antrian Q akan berisi elemen yang terurut menaik, atau lebih dikenal dengan *priority queue*. Untuk lebih jelasnya, urutan langkah yang harus ditempuh secara umum adalah sebagai berikut:

1. Masukkan lantai *Start* ke dalam antrian Q, jika lantai *Start* juga adalah lantai *Finish,* maka solusi telah ditemukan.
2. Jika antrian Q kosong, maka tidak ditemukan solusi.
3. Jika antrian Q tidak kosong, maka perluas elemen pertamanya, yaitu elemen yang memiliki ongkos terkecil. Jika ada elemen dengan nilai ongkos terkecil sama, maka dipilih salah satu secara sembarang.
4. Jika lantai yang akan dibangkitkan merupakan lantai *Finish*, maka solusi ditemukan. Jika bukan,

maka akan disisipkan lantai-lantai anak-anaknya, yaitu lantai di atas, kanan, kiri, dan bawah dari lantai tersebut, dengan urutan ongkos menaik pula. Lantai yang telah dibangkitkan anak-anaknya dihapus dari antrian Q. Jika lantai anak (lantai yang akan dibangkitkan) merupakan tembok, atau sudah pernah dilalui oleh pembangkitan lantai lainnya, maka lantai anak tersebut tidak dimasukkan ke dalam antrian Q.

5. Ulangi langkah ke-2.

Tinjau permasalahan berikut ini:



Gambar 2.3 Contoh Kasus Pemilihan Pembangkitan Simpul

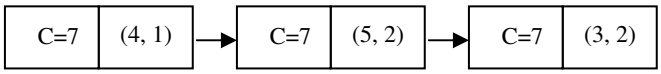
Pada langkah ketiga, uraian langkah di atas mengungkapkan bahwa pemilihan pembangkitan lantai jika ada yang bernilai (*cost*) sama dapat dilakukan secara sembarang. Tidak ada aturan yang pasti mengenai lantai mana yang harus terlebih dahulu dibangkitkan anak-anaknya. Ini berarti ada dua pilihan dalam membangkitkan lantai, jika ada lantai yang memiliki *cost* sama:

1. Membangkitkan anak-anak dari lantai yang baru masuk dari antrian Q.
2. Membangkitkan anak-anak dari lantai yang lebih dahulu masuk ke dalam antrian Q.

Apakah kedua cara tersebut akan menghasilkan lintasan yang optimal? Permasalahan ini akan dibahas pada uraian berikut ini, dengan mengacu permasalahan pada gambar 2.3:

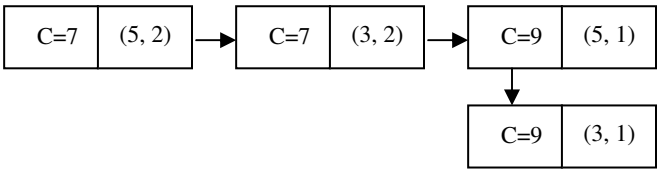
Cara 1: Membangkitkan anak-anak dari lantai yang baru masuk antrian Q

Untuk memudahkan dalam memperlihatkan urutan pembangkitan lantai, struktur antrian Q yang merupakan *priority queue* akan dijabarkan dalam bentuk *list* sederhana. List ini adalah list yang disusun sehingga elemen dengan nilai *cost* terkecil diletakkan di paling depan, dan elemen list terurut menaik.



Gambar 2.4 List *Priority Queue Untuk Gambar 2.3*

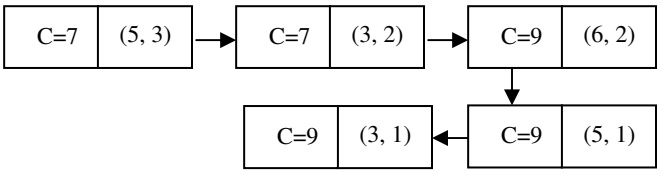
Selanjutnya, akan dibangkitkan lantai anak dari elemen pertama *list*, yaitu lantai (4, 1), dan lantai (4, 1) dihilangkan dari *list*:



Gambar 2.5 Pembangkitan anak-anak lantai (4, 1)

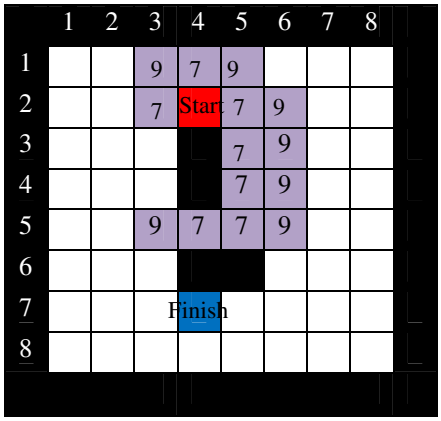
Permasalahan timbul saat akan membangkitkan anak dari lantai (5, 2). Anak dari lantai (5, 2), yaitu lantai (5,3) memiliki *cost* 7, sama dengan lantai (3,2) yang telah terdapat dalam *list.* Sesuai dengan perjanjian cara1, maka lantai (5,3) akan ditaruh sebagai elemen pertama dalam *list*, dan akan diperluas terlebih dahulu.

Keadaan *list* pada saat pembangkitan anak dari lantai (5, 2) dapat dilihat pada gambar berikut:



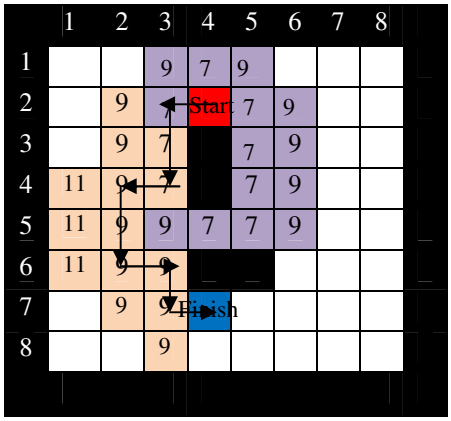
Gambar 2.6 Pembangkitan anak-anak (5, 2)

Jika cara ini kita terapkan terus-menerus, maka urutan pembangkitan anak suatu waktu akan mencapai tahap berikut ini:



Gambar 2.7 Urutan Pembangkitan Lantai dengan Cara 1

Sebagai akibatnya, lantai (3, 2) akan diperluas belakangan, setelah perluasan sampai di lantai (3, 5), dengan *cost* 9. Hal ini menjadi tidak menguntungkan karena robot tidak akan berjalan dalam lintasan yang paling minimum, diakibatkan perkembangan lantai yang tidak simetris, sehingga pembangkitan lantai dan jalur yang dipilih dapat dilihat pada gambar berikut:



Gambar 2.8 Lintasan yang Dipilih oleh Cara 1

Lintasan yang dipilih menjadi tidak optimal, karena jalur yang diambil oleh robot berkelok dan tidak mengambil lantai (3, 5), yang seharusnya diambil sehingga lintasan menjadi optimal.

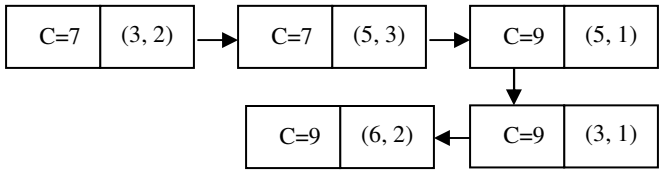
Secara lengkap, robot akan melewati lantai-lantai (3, 2), (3, 3), (3, 4), (2, 4), (2, 5), (2, 6), (3, 6), (3, 7), (4, 7) Total jumlah langkah yang terjadi adalah 9 langkah.

Cara 2: Membangkitkan anak-anak dari lantai yang lebih dahulu masuk ke dalam antrian Q.

Sekarang akan diuraikan cara pembangkitan lantai, apabila lantai yang terlebih dahulu masuk ke dalam antrian akan dibangkitkan terlebih dahulu anak-anaknya.

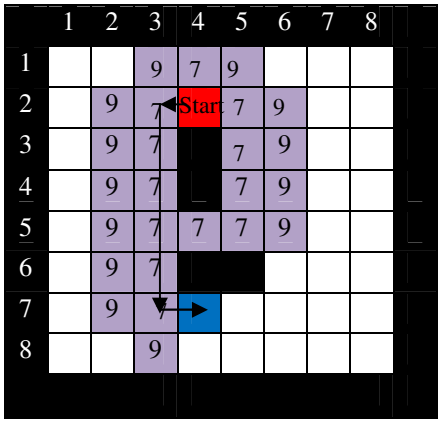
Hampir sama seperti cara 1, namun pada saat pembangkitan anak dari lantai (5, 2), yaitu lantai (5, 3) yang memiliki *cost* 7, *cost* yang sama seperti lantai (3, 2), akan tetapi kali ini lantai (5, 3) akan disisipkan di dalam *list* setelah lantai (3, 2), yang telah dimasukkan ke dalam *list* terlebih dahulu.

Keadaan list pada saat pembangkitan anak dari lantai (5, 2) dapat dilihat pada ilustrasi berikut



Gambar 2.9 Pembentukan *List* dengan Cara 2

Sehingga bila kita tinjau dari gambar labirin, maka urutan pembentukannya adalah sebagai berikut:



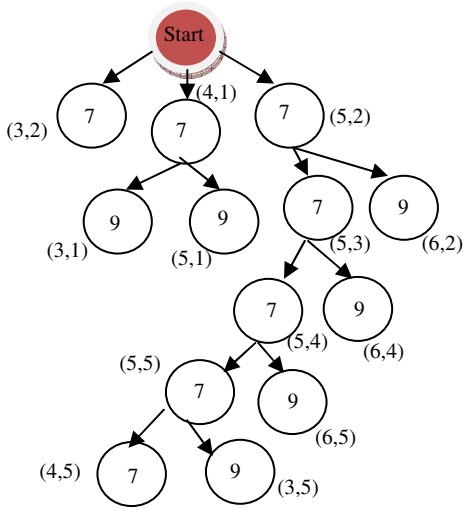
Gambar 2.10 Pembangkitan Lantai Menggunakan Cara 2

Lintasan yang dipilih sekarang melewati lintasan yang optimal. Secara lengkap robot akan melewati lantai-lantai (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (4, 7). Total jumlah langkah yang terjadi adalah 7 langkah.

3. PEMBAHASAN

Terjadi hal yang menarik di sini. Dengan menggunakan cara 1, didapatkan bahwa total jumlah langkah yang diperlukan untuk mencapai finish adalah 9 langkah, sedangkan total jumlah yang diperlukan bila menggunakan cara 2 adalah 7 langkah.

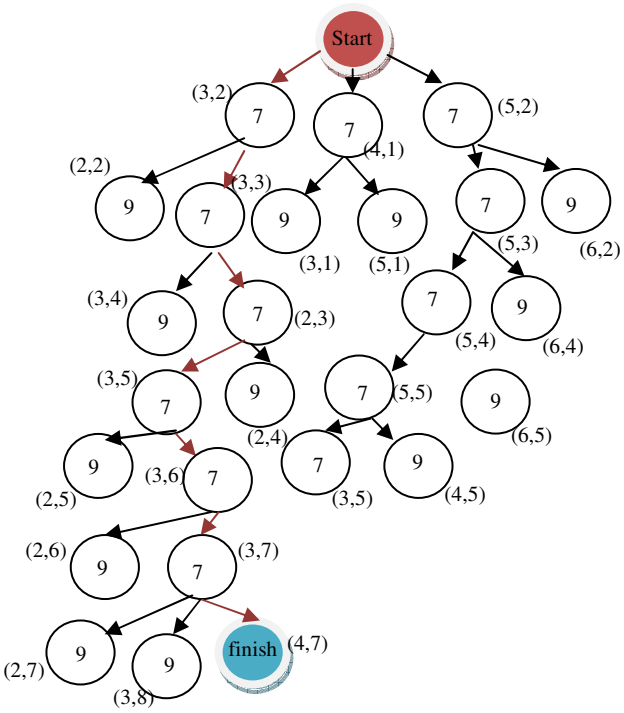
Cara 1 akan membangkitkan anak-anak dari lantai yang baru masuk ke dalam antrian dan memiliki *cost* terkecil, sehingga urutan pembangkitan seperti DFS, yaitu memperluas salah satu lantai sampai mendekati finish. Struktur pohonnya sebelum perluasan lantai (3,2):



Gambar 3.1 Pohon Pembentukan Cara 1 Sebelum (3, 2) Diperluas

Pada gambar 3.1 dapat kita lihat pohon yang terbentuk memperluas anak paling kanan terlebih dahulu. Hal ini akan mengakibatkan lantai (3,5) sudah menjadi simpul dari pohon pembentukan, yang akan berpengaruh terhadap pembangkitan anak lantai (3,2) nantinya. Hal inilah yang patut dihindari agar robot dapat mencari lintasan yang terpendek.

Sedangkan pada cara 2, lantai yang sudah masuk terlebih dahulu ke dalam antrian Q akan diperluas telebih dahulu, hal ini akan berakibat terbentuk pohon yang seimbang, antara perluasan lantai (3,2) dan lantai (5,2). Sebagian struktur pohonnya dapat dilihat pada gambar berikut:



Gambar 3.2 Pembentukan Pohon Cara 2

Dapat kita lihat pada pembentukan pohon dengan cara 2, jalan yang ditempuh menjadi optimal, yaitu 7 langkah, daripada membangkitkan lantai anak yang baru masuk ke dalam antrian Q terlebih dahulu.

4. KESIMPULAN

Dalam menggunakan algoritma *Branch and Bound*, maka sangat disarankan untuk memperhatikan urutan pembangkitan anak, walaupun pada saat ada lantai yang mempunyai *cost* sama, diutamakan untuk membangkitkan lantai yang sudah terlebih dahulu terdapat dalam antrian Q terlebih dahulu, sehingga arah pembentukan pohonnya menjadi berimbang, dan mengantisipasi kemungkinan solusi yang didapatkan tidak optimal, terutama dalam kasus pencarian jalur terpendek dengan menggunakan algoritma *Branch and Bound.*

REFERENSI

[1] R. Munir, "Diktat Kuliah IF2251 Strategi Algoritmik", Program Studi Teknik Informatika ITB, 2006.

[2] http://theory.stanford.edu/~amitp/GameProgramming/ Heuristics.html (tanggal akses: 17 Mei 2008)

[3]