Algoritma Pencarian A* dengan Fungsi Heuristik Jarak Manhattan

Puanta Della Maharani Riyadi - 13507135

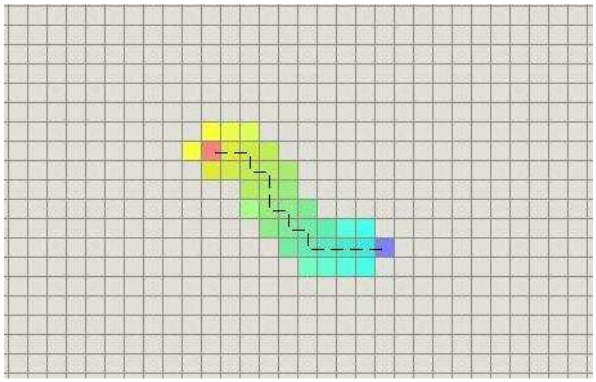
Teknik Informatika Institut Teknologi Bandung
Jl. Ganesha no. 10, Bandung
If17135@students.if.itb.ac.id

ABSTRAK

Permasalahan mencari dua titik terdekat sudah sering dijumpai dalam kehidupan sehari-hari. Dalam dunia komputasi, sangat banyak algoritma yang dapat digunakan untuk mencari 2 pasang titik yang terdapat pada suatu matriks (peta). Salah satu dari algoritma pencari pasangan titik terdekat itu ialah algoritma A*. Algoritma ini sangat mirip dengan algoritma BFS, namun ia memiliki fungsi heuristik tambahan. Algoritma A* juga terkenal sebagai algoritma yang mangkus dan pasti memberikan hasil yang optimal. Keberhasilan dari algoritma A* dalam memecahkan puzzle tak lepas dari penggunaan fungsi heuristik didalamnya, bahkan sifat (kelakuan) dari algoritma A* ini bisa berubah-rubah tergantung fungsi heuristik yang digunakannya. Fungsi heuristik ini sangat berpengaruh terhadap kecepatan kinerja maupun penggunaan sumber daya memori dari algoritma A*. Disini akan diuji coba jenis fungsi heuristik jarak Manhattan. Dari hasil pengujian itu akan dapat kita lihat kinerja A* dengan suatu fungsi heuristik tertentu. Pengujian algoritma A* akan dilakukan dengan 2 matriks berbeda.

Kata kunci: Algoritma A*, Heuristik, Jarak Manhattan, BFS, DFS, labirin.

1. PENDAHULUAN



Gambar 1. Hasil algoritma pencarian A*

A* (dibaca "A bintang"/"A star") adalah algoritma pencarian graf/pohon yang mencari jalur dari satu titik awal ke sebuah titik akhir yang telah ditentukan. Algoritma A* menggunakan pendekatan heuristik h(x) yang memberikan peringkat ke tiap-tiap titik x dengan cara memperkirakan rute terbaik yang dapat dilalui dari titik tersebut. Setelah itu tiap-tiap titk x tersebut dicek satu-persatu berdasarkan urutan yang dibuat dengan pendekatan heuristik tersebut. Maka dari itulah algoritma A* adalah contoh dari best-first search.

Algoritma ini pertama kali ditemukan pada tahun 1968 oleh Peter Hart, Nils Nilsson dan Bertram Raphael. Dalam tulisan mereka, algoritma ini dinamakan algoritma A. Penggunaan algoritma ini dengan fungsi heuristik yang tepat dapat memberikan hasil yang optimal, maka algoritma inipun disebut A*.

1.1 Deskripsi Algoritma

Berikut diberikan pseudo-code dari algoritma A*:

```
function A*(start,goal)
     closedset := the empty set
% The set of nodes already evaluated.
     openset := set containing the
initial node % The set of tentative
nodes to be evaluated.
     g_score[start] := 0
% Distance from start along optimal
path.
     h_score[start] :=
heuristic_estimate_of_distance(start,
goal)
     f_score[start] := h_score[start]
% Estimated total distance from start
to goal through y.
     while openset is not empty
         x := the node in openset
having the lowest f_score[] value
         if x = goal
             return
reconstruct_path(came_from,goal)
         remove x from openset
         add x to closedset
         foreach y in
neighbor_nodes(x)
```

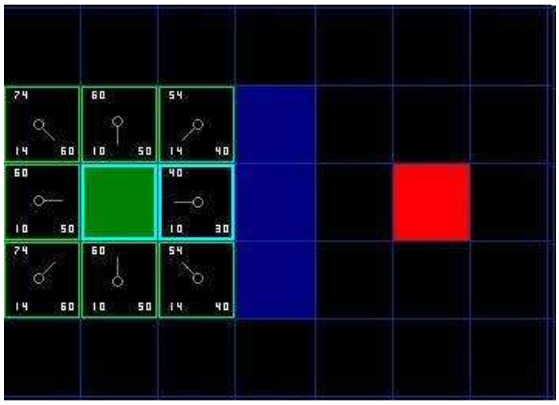
```
            if y in closedset
                continue
            tentative_g_score :=
g_score[x] + dist_between(x,y)

            if y not in openset
                add y to openset

                tentative_is_better
:= true
            elseif tentative_g_score
< g_score[y]
                tentative_is_better
:= true
            else
                tentative_is_better
:= false
            if tentative_is_better =
true
                came_from[y] := x
                g_score[y] :=
tentative_g_score
                h_score[y] :=
heuristic_estimate_of_distance(y,
goal)
                f_score[y] :=
g_score[y] + h_score[y]
    return failure

 function
reconstruct_path(came_from,current_no
de)
    if came_from[current_node] is
set
        p =
reconstruct_path(came_from,came_from[
current_node])
        return (p + current_node)
    else
        return the empty path
```

Sebenarnya, Depth-first search(DFS) dan breadth-first search(BFS) adalah dua kasus khusus dari algoritma A*. Algoritma Dijkstra, salah satu BFS, adalah kasus khusus dari A* dimana h(x) = 0 untuk semua nilai x. Untuk DFS, ciptakan suatu counter global C yang diinisialisasi dengan nilai yang sangat besar. Pada setiap langkahnya, periksa sebuah titik, lalu berikan nilai C ke semua titik yang bertetangga dengan titik tadi. Setelah tiap-tiap pemberian nilai , kurangi counter C dengan 1. Jadi semakin awal sebuah titik diproses, semakin tinggi nilai h(x) yang dimilikinya.

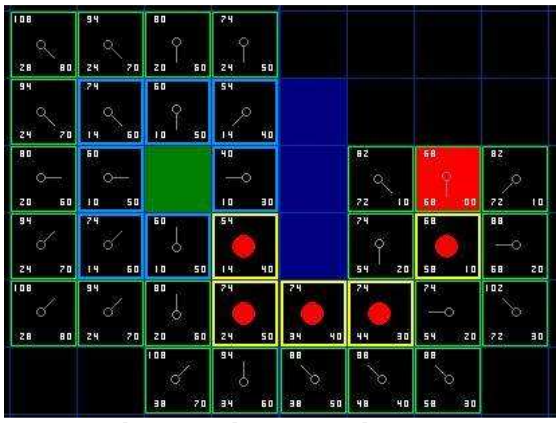
A* menyimpan sebuah himpunan solusi parsial, jalur yang diambil yang berawal dari titik awal, disimpan dalam sebuah antrian berprioritas(priority queue). Prioritas yang diberikan pada suatu jalur ditentukan oleh fungsi f(x) = g(x) + h(x).

Di sini, g(x) adalah ongkos yang diperlukan sampai ke titik x sejauh ini, h(x) adalah perkiraan heuristik dari ongkos minimal untuk meraih titik tujuan dari titik x. Singkat kata, ongkos adalah jarak yang telah ditempuh sejauh ini, dan panjang garis lurus antara titik x dengan titik akhir adalah perkiraan heuristiknya. Semakin rendah nilai f(x), semakin tinggi prioritasnya.

Di sini, suksesor(p) mengembalikan sebuah himpunan titik yang bertetangga dengan p. Diasumsikan bahwa antrian disini menjaga urutan anggotanya dengan nilai f secara otomatis.



Gambar 2. Arah gerak pencarian algoritma A*



Gambar 3. Hasil pencarian algoritma A*

Dalam himpunan himpunan_tertutup, semua titik akhir dari p (titik dimana masih terdapat jalur) disimpan, untuk menghindari pengulangan dan pemutaran. Antrian yang digunakan disini juga biasa disebut sebagai himpunan terbuka. Himpunan tertutup bisa saja dihilangkan (yang akan menghasilkan algoritma pencarian pada pohon) apabila sebuah solusi(jalur) telah dijamin keberadaannya, atau apabila fungsi suksesor telah disesuaikan agar tidak menerima titik yang sudah pernah diperiksa.

1.2 Sifat

Seperti BFS, A* akan selalu menemukan sebuah solusi, jika memang ada. Apabila fungsi heuristik h dapat diterima, yang berarti nilai h tidak akan pernah melebihi ongkos minimal untuk meraih tujuan yang sebenarnya, maka A* sendiri akan dapat diterima (atau dapat disebut optimal) apabila himpunan tertutup tidak digunakan. Apabila digunakan sebuah himpunan tertutup, maka h harus monoton (atau konsisten) agar A* dapat menjadi optimal. Ini berarti fungsi heuristik tidak akan pernah berlebihan dalam menghitung ongkos dari sebuah titik ke titik tetangganya. Secara formal, unutuk semua jalur x,y dimana y adalah suksesor dari x:

$$h(x) \leq g(y) - g(x) + h(y) \qquad (1)$$

A* juga dapat dijamin keoptimalannya untuk sembarang heuristikh, yang berarti bahwa tidak ada satupun algoritma lain yang mempergunakan heuristik yang sama akan mengecek lebih sedikit titik dari A*, kecuali ketika ada beberapa solusi parsial dimana h dapat dengan tepat memprediksi ongkos jalur minimal.

1.3 Kompleksitas

Kompleksitas waktu dari A* sangat bergantung dari heuristik yang digunakannya. Pada kasus terburuk, jumlah titik yang diperiksa berjumlah eksponensial terhadap panjang solusi (jalur terpendek), tetapi A* akan memiliki kompleksitas waktu polinomial apabila fungsi memenuhi kondisi berikut:

$$|h(x) - h^*(x)| \leq O(\log h^*(x) \qquad (2)$$

dimana h* adalah heuristik optimal, yaitu ongkos sebenarnya dari jalur x ke tujuan. Dengan kata lain, galat dari h tidak akan melaju lebih cepat dari logaritma dari heuristik optimal h* yang memberikan jarak sebenarnya dari x ke tujuan (Russel dan Norvig 2003, p. 101).

Penggunaan memori dari A* bahkan lebih bermasalah dari kompleksitas waktunya. Beberapa varian dari A* telah dikembangkan untuk mengatasi masalah ini, termasuk diantaranya Iterative-Deepening-A*, Memory-Bounded-A* (MA*) dan Simplified-Memory-Bounded-A* (SMA*) dan Recursive-Best-First-Search (RBFS). RBFS juga akan memberikan hasil yang optimal apabila heuristiknya dapat diterima.

2. FUNGSI HEURISTIK UNTUK A*

Seperti yang telah disebutkan diatas, fungsi heuristik sangat berpengaruh terhadap kelakuan algoritma A*:

- Apabila h(n) selalu bernilai 0, maka hanya g(n) yang akan berperan, dan A* berubah menjadi Algoritma Dijkstra, yang menjamin selalu akan menemukan jalur terpendek.
- Apabila h(n) selalu lebih rendah atau sama dengan ongkos perpindahan dari titik n ke tujuan, maka A* dijamin akan selalu menemukan jalur terpendek. Semakin rendah nilai h(n), semakin banyak titik-titik yang diperiksa A*, membuatnya semakin lambat.
- Apabila h(n) tepat sama dengan ongkos perpindahan dari n ke tujuan, maka A* hanya akan mengikuti jalur terbaik dan tidak pernah memeriksa satupun titik lainnya, membuatnya sangat cepat. Walaupun hal ini belum tentu bisa diaplikasikan ke semua kasus, ada beberapa kasus khusus yang dapat menggunakannya.
- Apabila h(n) kadangkala lebih besar dari ongkos perpindahan dari n ke tujuan, maka A* tidak menjamin ditemukannya jalur terpendek, tapi prosesnya cepat.
- Apabila h(n) secara relatif jauh lebih besar dari g(n), maka hanya h(n) yang memainkan peran, dan A* berubah menjadi BFS.

Heuristik yang paling umum digunakan adalah jarak Manhattan. Fungsi heuristik ini hanya akan menjumlahkan selisih nilai x dan nilai y dari dua buah titik. Heuristik ini dinamakan Manhattan mungkin karena di kota Manhattan di Amerika, jarak dari dua lokasi umumnya dihitung dari blok-blok yang harus dilalui saja dan tentunya tidak bisa dilintasi secara diagonal. Perhitungannya dapat ditulis sebagai berikut:

$$h(n) = \text{abs}(n.x - \text{tujuan}.x) + \text{abs}(n.y - \text{tujuan}.y) \qquad (3)$$

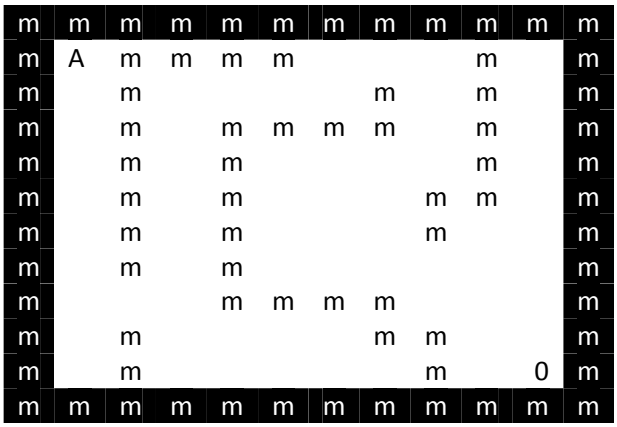
Hal lain yang harus diperhatikan adalah seberapa cepat
fungsi heuristik dapat dikomputasi. Selalu akan ada trade-off antara akurasi dari fungsi heuristik dan waktu yang dibutuhkannya untuk mengomputasinya. Nampaknya bagus jika fungsi heuristik yang digunakan sangat akurat, dilihat dari berbagai macam percobaan bahwa jika heuristik yang digunakan sempurna maka A* akan selalu melewati jalur yang tepat dan akan selalu memberikan optimum global. Namun, heuristik yang sempurna semacam itu tidak ada (dan tidak akan pernah ada), dan bahkan untuk mendekatinya saja akan memerlukan tambahan komputasi yang tidak ringan.

Seringkali dalam aplikasinya heuristik yang memberikan hasil yang sangat akurat namun lambat, kurang disenangi dibanding heuristik yang tidak begitu optimal namun memberikan hasil dengan cepat. Maka dari itu, pemilihan heuristik sangat bergantung pada tujuan penggunaan A*. Jika hasil yang dibutuhkan adalah optimum global, maka fungsi heuristik yang digunakannya haruslah "sempurna", sedang jika yang dibutuhkan adalah hasil yang cepat dan tidak harus jalur

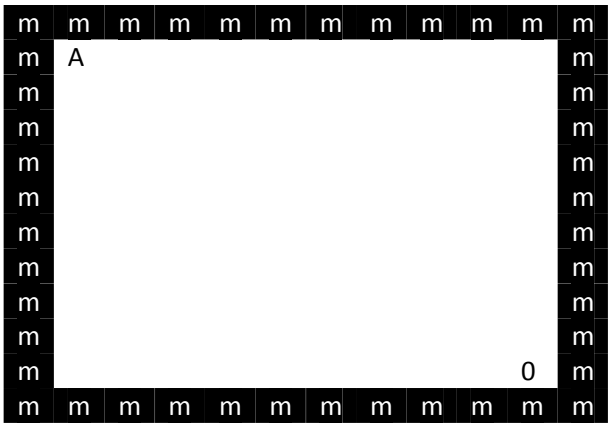
terpendek, maka lebih bijak menggunakan heuristik yang lebih ringan.

3. PENGAPLIKASIAN ALGORITMA A*

Algoritma A* akan diaplikasikan pada dua buah peta yang berbeda disini. Peta pertama adalah suatu matriks yang berisi tembok-tembok semi labirin, dengan tampilan seperti di gambar 2. Untuk selanjutnya matriks tersebut akan disebut sebagai Peta 1 untuk memudahkan. Peta kedua merupakan suatu matriks kosong yang dikelilingi tembok seperti yang tertera pada gambar 3, yang berikutnya akan disebut Peta 2. Semua matriks yang digunakan disini memiliki ukuran 12x12, dengan hitungan untuk tembok, maka matriks yang digunakan menjadi 10x10 saja.



Gambar 2. Peta 1



Gambar 3. Peta 2

Karakter A dan 0 dipergunakan untuk menunjukkan lokasi titik awal(A) dan titik akhir(0), sedangkan 'm' adalah tembok (penghalang). Tujuan dalam percobaan kali ini bukan untuk menemukan suatu optimum global dari suatu peta, melainkan hanya gambaran kasar dari sangat bergantungnya kecepatan algoritma A* dari fungsi heuristik yang digunakannya.

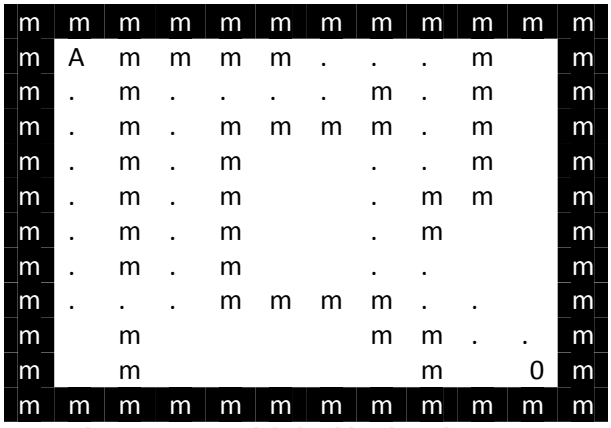
Perhitungan kemangkusan dari suatu heuristik disini menggunakan banyaknya titik yang diperiksa sebelum didapat suatu hasil akhir. Perbandingan dari segi waktu tidak ditampilkan disini karena hasilnya bisa sangat bervariasi tergantung pada kemampuan komputasi komputer yang digunakan, selain itu jumlah titik yang diperiksa juga selalu berbanding lurus dengan waktu.

Hasil yang akan didapat mungkin saja lebih besar dari banyak titik yang berada pada peta yang digunakan (10x10). Hal ini dikarenakan algoritma A* bisa saja mengganti nilai g(n) dari suatu titik n, jika terdapat jalur yang lebih dekat untuk meraih titik tersebut.

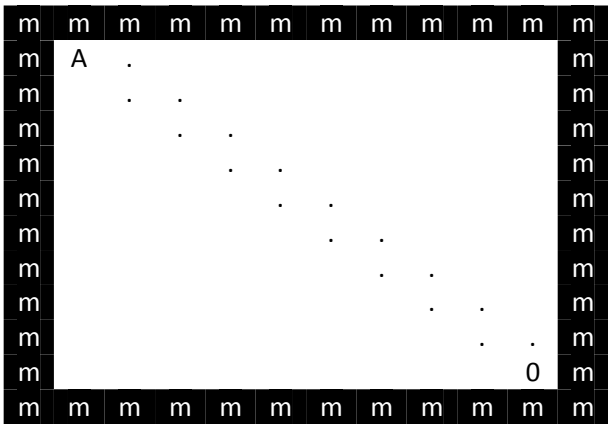
3.1 Hasil Percobaan

Peta 1

- Tanpa Heuristik: 124 titik yang diperiksa sebelum titik tujuan ditemukan.
- Jarak Manhattan: 175 titik yang diperiksa sebelum titik tujuan ditemukan.



Gambar 4. Peta 1 setelah diaplikasikan algoritma A*



Gambar 5. Peta 2 setelah diaplikasikan algoritma A*

Peta 2
• Tanpa Heuristik: 344 titik yang diperiksa sebelum titik tujuan ditemukan.
• Jarak Manhattan: 344 titik yang diperiksa sebelum titik tujuan ditemukan.

3.2 Analisis Hasil

Jika melihat hasil yang didapatkan diatas maka satu hal pasti yang dapat kita lihat adalah betapa berpengaruhnya fungsi heuristik terhadap algoritma A* ini, walaupun dalam kasus ini hasil yang didapat jika menggunakan fungsi heuristik malah lebih buruk.

4. KESIMPULAN

Seperti yang telah disampaikan di awal, algoritma A* ini benar-benar terpengaruh oleh fungsi heuristik yang digunakannya. Hasil yang didapat jika kita menggunakan fungsi heuristik yang tepat benar-benar dapat mengoptimalisasi waktu yang dibutuhkan untuk mencari solusi. Selain ini, dapat dibuktikan untuk kasus ini ketiga fungsi heuristik jarak Manhattan, jarak Euclid dan jarak Euclid yang dikuadratkan sama-sama berpengaruh buruk terhadap kinerja algoritma A*. Mungkin saja pendekatan-pendekatan fungsi heuristik lainnya bisa membantu meningkatkan kinerja algoritma A* ini.

REFERENSI

[1] A* search algorithm, http://en.wikipedia.org/wiki/A_search_algorithm, Wikipedia, tanggal akses 2 januari 2010 pukul 13.00 WIB.
[2] www. policyalmanac.org/games/
Tanggal akses 2 Januari 2010 pukul 13.00
[3] theory.stanford.edu/~amitp/
Tanggal akses 2 Januari 2010 pukul 13.00
[4]Russell, S. J., Norvig, P, Artificial Intelligence: A Modern Approach, 2003.
[3] Munir, Rinaldi, Diktat Kuliah IF2251 Strategi Algoritmik, Penerbit ITB, 2007.