BAB III

ALGORITMA *GREEDY* DAN ALGORITMA A*

3.1 *Best First Search*

Sesuai dengan namanya, *best-first search* merupakan sebuah metode yang membangkitkan simpul dari sebuah simpul sebelumnya (yang sejauh ini terbaik di antara semua simpul yang pernah dibangkitkan). Penentuan simpul terbaik dilakukan dengan menggunakan sebuah fungsi yang disebut **fungsi evaluasi *f(n)*** (Russel and Norvig, 1995). Pada *best-first search* jika suksesor digunakan, maka dikatakan algoritma mengembangkan simpul tersebut. Setiap sebuah simpul dikembangkan, algoritma akan menyimpan setiap suksesor simpul *n* sekaligus dengan harga (*cost*) dan petunjuk pendahulunya yang disebut "*parent*". Algoritma akan berakhir pada simpul tujuan, dan tidak ada lagi pengembangan simpul. Fungsi evaluasi pada *best-first search* dapat berupa informasi biaya perkiraan dari suatu simpul menuju ke simpul tujuan atau gabungan antara biaya sebenarnya dan biaya perkiraan tersebut. Biaya perkiraan dapat diperoleh dengan menggunakan suatu fungsi yang disebut fungsi *heuristic*. Pada strategi *best first search, cost* sebenarnya yaitu dari simpul awal ke simpul *n*, dinotasikan dengan *g(n)* dan fungsi *heuristic* yang digunakan yaitu perkiraan/etimasi nilai dari simpul *n* ke simpul tujuan dinotasikan dengan *h(n)*. Algortima yang menggunakan metode *best-first search*, yaitu:

- *Greedy Best-First Search*
- Algoritma A*

Berikut akan diberikan beberapa istilah yang yang sering digunakan pada metode *best first search*:

- *Start node* adalah sebuah terminologi untuk posisi awal sebuah pencarian.
- *Current Node* adalah simpul yang sedang dijalankan (yang sekarang) dalam algortima pencarian jalan terpendek.
- Kandidat (*successor)* adalah simpul simpul yang berajasen dengan current *node*, dengan kata lain simpul-simpul yang akan diperiksa berikutnya.
- Simpul (*node*) merupakan representasi dari area pencarian
- *Open list* adalah tempat menyimpan data simpul yang mungkin diakses dari *starting node* maupun simpul yang sedang dijalankan.
- *Closed list* adalah tempat menyimpan data simpul yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan.
- *Goal node* yaitu simpul tujuan.
- *Parent* adalah *current node* dari suksesor/kandidat.

3.2 *Greedy Best First Search*

Salah satu algoritma yang termasuk kedalam kategori *informed search* adalah *Greedy Best First Search* yang dikenal juga dengan *Greedy Search*. Secara harfiah *greedy* artinya rakus atau tamak, sifat yang berkonotasi negatif. Sesuai dengan arti tersebut, prinsip *greedy* adalah mengambil keputusan yang dianggap terbaik hanya untuk saat itu saja yang diharapkan dapat memberikan solusi terbaik secara keseluruhan. Oleh karena itu, pada setiap langkah harus

dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya.

Greedy Best First Search seperti halnya algoritma yang yang menggunakan strategi *best-first search* lainnya mempunyai sebuah fungsi yang menjadi acuan kelayakan sebuah simpul yaitu fungsi evaluasi *f(n).* Pada *Greedy Best First Search* fungsi evaluasi tidak bergantung pada *cost* sebenarnya, tetapi hanya tergantung pada fungsi *heuristic* itu sendiri. Jika pada algoritma Dijkstra pencarian yang dilakukan bergantung pada *cost* sebenarnya dari sebuah simpul yaitu *g(n),* pada *Greedy Best First Search* fungsi evaluasi hanya bergantung pada fungsi *heuristic h(n)* yang mengestimasikan arah yang benar, sehingga pencarian jalur dapat berlangsung dengan sangat cepat. Secara matematis fungsi evaluasi pada *greedy search* diberikan oleh :

$$f(n) = h(n)$$

dengan:

g(n) = estimasi biaya dari simpul *n* ke simpul tujuan *(goal node)*

Berikut langkah-langkah pencarian lintasan terpendek yang dilakukan *Greedy Best-First Search* :

- Masukkan simpul awal ke dalam *Open List.*
- *Open* berisi simpul awal dan *Closed List* masih kosong.
- Masukkan simpul awal ke *Closed List* dan suksesornya pada *Open List.*
- Ulangi langkah berikut sampai simpul tujuan ditemukan dan tidak ada lagi simpul yang akan dikembangkan.

- Hitung nilai *f* simpul-simpul yang ada pada *Open List,* ambil simpul terbaik (*f* paling kecil).
- Jika simpul tersebut sama dengan simpul tujuan, maka sukses
- Jika tidak, masukkan simpul tersebut ke dalam *Closed*
- Bangkitkan semua suksesor dari simpul tersebut
- Untuk setiap suksesor kerjakan :
    - Jika suksesor tersebut belum pernah dibangkitkan, evaluasi suksesor tersebut, tambahkan ke *Open*, dan catat "*parent*"-nya.
    - Jika suksesor tersebut sudah pernah dibangkitkan, ubah *parent*-nya jika jalur melalui *parent* ini lebih baik daripada jalur melalui *parent* yang sebelumnya. Selanjutnya, perbarui biaya untuk suksesor tersebut.

3.3 Algoritma A* (*A Star*)

Terdapat banyak algoritma pencarian lintasan terpendek, Algoritma Dijsktra merupakan salah satu dari algoritma tersebut. Dengan menggunakan fungsi biaya *g(n)* setiap simpul, algoritma Dijkstra memeriksa kelayakan biaya yang diperlukan untuk mencapai suatu simpul dari sebuah simpul lain. Proses ini dilakukan berulang sampai simpul tujuan diperiksa.

Algoritma Dijkstra memang menjamin didapatkannya jalur optimal, tetapi algoritma ini mempunyai kelemahan. Pemeriksaan simpul akan dilakukan ke segala arah yang dimungkinkan yang pada akhirnya seluruh simpul pada sebuah graf akan diperiksa. Hal ini menyebabkan algoritma ini bekerja dengan lambat

sehingga waktu yang diperlukan untuk menemukan solusi akan semakin besar pula.

Algoritma A* adalah algoritma yang menggabungkan algoritma Dijkstra dan algoritma *Greedy Best First Search*. Selain menghitung biaya yang diperlukan untuk berjalan dari simpul satu ke simpul lainnya, algoritma A* juga menggunakan fungsi *heuristic* untuk memprioritaskan pemeriksaan simpul - simpul pada arah yang benar, sehingga algoritma A* mempunyai efisiensi waktu yang baik dengan tidak mengorbankan perhitungan biaya sebenarnya.

3.3.1 Sejarah Algorima A*

Penggunaan informasi *heuristic* untuk meningkatkan efisiensi pencarian telah dipelajari dalam berbagai bidang. Penggunaan fungsi evaluasi pada masalah pencarian pada graf dikemukakan oleh Lin (1965) yang digunakan pada masalah *Traveling Salesman Problem (TSP).* Doran dan Michie (1966) merumuskan dan mencoba dengan algoritma *best-first search* yang menggunakan perkiraan jarak dari *current node* ke simpul tujuan. Hart, Nilsson dan Raphael memperkenalkan penggunaan sebuah algoritma dalam masalah optimasi yaitu algoritma A* (A star). Versi *bi-directional* algoritma A*, yang secara bersamaan mencari dari simpul awal dan simpul tujuan diperkenalkan oleh Pohl (1971), yang selanjutnya diteliti oleh de Champeaux dan Sint (1977).

3.3.2 Algoritma A* Dalam Pencarian Rute Terpendek

Beberapa stategi *best first search* berbeda hanya pada bentuk fungsi evaluasi yang digunakan. Pada strategi *best first search,* simpul yang terpilih untuk dikembangkan adalah simpul dengan nilai fungsi evaluasi terendah dan

ketika dua lintasan mengarah pada simpul yang sama, simpul dengan nilai paling besar akan dibuang. Secara matematis, nilai fungsi evaluasi *heuristic* sebuah simpul pada algoritma A* diberikan oleh *:*

$$f(n) = h(n) + h(n)$$

dengan,

g(n) = biaya dari simpul awal *(start node)* ke simpul *n.*

h(n) = estimasi biaya dari simpul *n* ke simpul tujuan *(goal node)*

Algoritma A* menggunakan dua buah list yaitu *Open List* dan list *Closed List*. Seperti halnya *best-first search* yang lain kedua *list* mempunyai fungsi yang sama. Pada awalnya *Open List* hanya berisi satu simpul yaitu simpul awal dan *Closed List* masih kosong. Perlu diperhatikan setiap simpul akan menyimpan petunjuk "*parent*"-nya sehingga setelah pencarian berakhir lintasan juga akan didapatkan. Berikut adalah langkah-langkah algoritma A* (A *Star)* :

- Masukkan simpul awal ke *Open List.*
- Ulangi langkah berikut sampai pencarian berakhir.
  - Cari *node* n dengan nilai *f(n)* paling rendah, dalam *Open List*. Node ini akan menjadi *current node.*
  - Keluarkan *current node* dari *Open List* dan masukkan ke *Closed List*.
  - Untuk setiap suksesor dari *current node* lakukan langkah berikut :
    i. Jika sudah terdapat dalam *Closed List*, abaikan, jika tidak lanjutkan.

ii. Jika belum ada pada *Open List*, masukkan ke *Open List.* Simpan *current node* sebagai "*parent*" dari suksesor-suksesor ini. Simpan *cost* masing-masing simpul

iii. Jika sudah ada dalam *Open List*, periksa jika simpul suksesor ini mempunyai nilai lebih kecil dibanding suksesor sebelumya. Jika lebih kecil, jadikan sebagai *current node* dan ganti "*parent*" *node* ini.

- Walaupun telah mencapai simpul tujuan, jika masih ada suksesor yang memiliki nilai yang lebih kecil, maka simpul tersebut akan terus dipilih sampai bobotnya jauh lebih besar atau mencapai simpul akhir dengan bobot yang lebih kecil dibanding dengan simpul sebelumnya yang telah mencapai simpul tujuan.
- Pada setiap pemilihan simpul berikutnya, nilai *f(n)* akan dievaluasi, dan jika terdapat nilai *f(n)* yang sama maka akan dipilih berdasarkan nilai *g(n)* terbesar.

3.4 *Heuristic Best First Search*

Fungsi *heuristic h(n)* merupakan estimasi *cost* dari *n* ke simpul tujuan. sangat penting untuk memilih fungsi *heuristic* yang baik. Misalkan *h*(n)* merupakan *cost* sebenarnya dari simpul *n* ke simpul tujuan, maka pada algoritma A* terdapat beberapa kemungkinan yang terjadi pada pemilihan fungsi *heuristic* yang digunakan, yaitu (Amit Gaming):

- Jika $h(n) = 0$, sehingga hanya $g(n)$ yang yang terlibat maka A* akan bekerja seperti halnya algoritma Dijkstra.
- Jika $h(n) \leq h^*(n)$, maka A* akan mengembangkan titik dengan nilai paling rendah dan algoritma A* menjamin ditemukannya lintasan terpendek. Nilai $h(n)$ terendah akan membuat algoritma mengembangkan lebih banyak simpul. Jika $h(n) \leq h^*(n)$, maka $h(n)$ dikatakan *heuristic* yang *admissible*.
- Jika $h(n) = h^*(n)$, maka A* akan mengikuti lintasan terbaik dan tidak akan mengembangkan titik-titik yang lain sehingga akan berjalan cepat. Tetapi hal ini tidak akan terjadi pada semua kasus. Informasi yang baik akan mempercepat kinerja A*.
- Jika $h(n) \geq h^*(n)$, maka A* tidak menjamin pencarian rute terpendek, tetapi berjalan dengan cepat.
- Jika $h(n)$ terlalu tinggi relative dengan $g(n)$ sehingga hanya $h(n)$ yang bekerja maka A* berubah jadi *Greedy Best first search.*

Berikut beberapa *heuristic* yang biasa digunakan yaitu:

3.4.1 *Manhattan Distance*

Manhattan distance atau sering disebut *Taxicab Geometry*, *city block distance*, diperkenalkan oleh Hermann Minkowski pada abad ke-19. *Manhattan distance* merupakan *heuristic* standar. Fungsi *heuristic* ini digunakan untuk kasus dengan pergerakan pada peta hanya lurus (horisontal atau vertikal), tidak diperbolehkan pergerakan diagonal. *Manhattan distance* antara dua vektor p,q pada sebuah dimensi n adalah penjumlahan panjang proyeksi garis antara dua

objek. Secara formal perhitungan nilai *heuristic* untuk simpul ke-*n* menggunakan *Manhattan distance* adalah sebagai berikut :

$$d(p, q) = \|p - q\| = \sum_{i=1}^{n} [p_i - q_i]$$

Pada kasus dua dimensi dan pada peta geografis *Manhattan Distance* diberikan oleh*:*

$$h(n) = ((n.x\text{-}goal.x) + (n.y\text{-}goal.y))$$

Dengan,

h(n) = nilai *heuristic* untuk simpul n

n.x = nilai koordinat *x* dari simpul n

n.y = nilai koordinat *y* dari simpul n

x-goal = nilai koordinat *x* dari simpul tujuan

y-goal = nilai koordinat *y* dari simpul tujuan

3.4.2 *Euclidean distance*

Euclidean distance didefinisikan sebagai panjang dari garis lurus yang menghubungkan posisi dua buah objek. Secara logis diketahui bahwa jarak terpendek antara dua titik adalah garis lurus antara kedua titik tersebut. *Euclidean distance* digunakan jika proses dapat bergerak ke segala arah.

Ecuclidean distance antara titik p dan q adalah panjang segmen garis $\overline{pq}$.

Pada koordinat *Cartesian jika* $p=(p_1,p_2,\ldots,p_n)$ *dan* $q=(q_1,q_2,\ldots,q_n)$ merupakan dua buah titik pada ruang *n*, maka jarak antara kedua titik dari *p* ke *q* diberikan oleh *:*

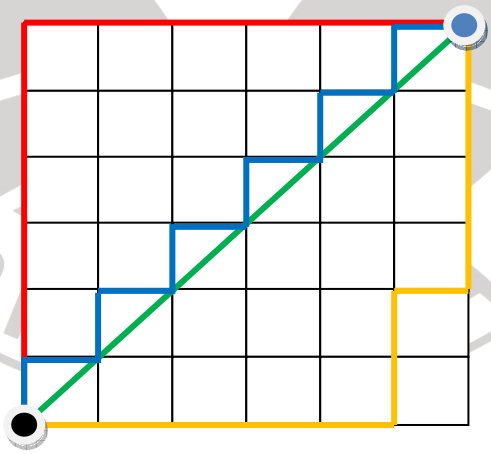
$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

Pada bidang datar, dua buah titik dengan koordinat *Cartesian*, (x_i,y_i) dan (x_j,y_j), *Euclidean Distance* diberikan oleh:

$$h(n) = dE(i,j) = \sqrt{[(xi - xj)2 + (yi - yj)2]}$$

Euclidean distance bertujuan untuk memprioritaskan *node-node* yang berada dekat garis lurus antara simpul awal dan simpul tujuan. Pendekatan ini dapat sangat membantu algoritma A* karena nilainya yang tidak pernah akan melebihi nilai sebenarnya. Namun pendekatan ini dapat tidak berpengaruh ataupun malah memperlambat kinerja algoritma A*.

Karena *Euclidean distance* lebih pendek dari Manhattan *distance* atau *diagonal distance* akan didapat lintasan terpendek, namun waktu yang dibutuhkan akan bertambah.



Gambar 3.1 *Eulidean Distance* dan *Manhattan Distance*

Gambar 3.1 memperlihatkan pendekatan *Euclidean distance* dan *Manhattan distance*. Merah, biru dan kuning mempunyai nilai yang sama yaitu 12

merupakan hasil *Manhattan distance* untuk rute yang sama dengan *Euclidean Distance*, warna hijau yang mempunyai panjang 6 x $\sqrt{2} \approx 8{,}48$ yang lebih kecil dari *Manhattan distance.*