Analisis Perbandingan Implementasi Algoritma *Shortest Path A** dan *Flood-Fill* pada Kompetisi *Micromouse*

Arvin Yustin
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517124@std.stei.itb.ac.id

Abstraksi - Micromouse Competition adalah kompetisi tahunan yang memperlombakan robot-robot mini untuk menyelesaikan jalur pada maze ukuran 16x16. Kompetisi Micromouse dimulai sejak tahun 1970-an, dan hingga sekarang terdapat beberapa kejuaraan tingkat regional, seperti Micromouse USA, All-Japan Micromouse Contest, IIT Kharagpur's Robotics Competitions.

Kata kunci – heuristik, robot, micromouse, A-star, flood-fill, priority queue, Manhattan distance, best first-search, Dijkstra, upper-bound

I. PENGENALAN

Dalam dunia nyata, banyak hal tidak dapat terlepas dalam pikiran kita, terutama dalam memikirkan cara menuju ke sebuah tempat. Ambillah contoh kita hendaknya pergi ke rumah kos seorang teman dan anggaplah teknologi masih belum mendukung GPS (*Global Positioning System*). Pertanyaan yang pertama kali muncul ialah 'bagaimana'. Bagaimana kita sampai tujuan (dalam hal ini rumah kos teman). Diberikan sebuah alamat jalan, dan nomor rumah, Kita hanya bisa mencocokkan nama jalan dan posisi samar rumah kos dari perkiraan letak nomor rumah. Pengetahuan akan perkiraan jalan dan letak nomor rumah itu merupakan landasan dari teori heuristik, dimana robot diberikan sebuah *clue* dalam pencariannya. Pada perkembangan algoritma *shortest path*, sering digunakan bantuan heuristik sehingga algoritma tersebut mempunyai "otak" pemroses apa yang benar, dan apa yang salah. Algoritma yang memakai sistem heuristik di antaranya ialah A-*star* dan *Flood-filling*. Algoritma ini banyak diimplementasikan ketika robot sudah mengetahui secara kasar dimensi atau karakteristik area yang akan menjadi lingkungannya

II. TEORI DASAR

A. Algoritma A*

Algoritma A* (dibaca *A-star*) ialah kombinasi konsep algoritma *Dijkstra* dan *Best First-Search*. Pada algoritma *Dijkstra*, hal yang diperhatikan hanyalah jarak simpul sedekat mungkin dengan titik awal. Sedangkan *Best First-Search* memperhatikan jarak heuristik antara titik sekarang dengan titik tujuan (jarak heuristik harus selalu *overestimate* karena permintaan untuk mendapatkan jarak minimum).

Algoritma A* dibantu oleh sebuah fungsi f(x) = g(x) + h(x), dimana g(x) merupakan jarak titik awal ke titik sekarang, dan h(x) jarak heuristik titik sekarang ke tujuan. Setiap pergerakan dari sebuah robot, fungsi f(x) diperhatikan sehingga selalu memberikan solusi optimal. Fungsi heuristik dihitung dari formula *Manhattan Distance*, yaitu banyak langkah menuju titik tujuan. Fungsi jenis ini idealnya digunakan jika gerak robot dibatasi atas-bawah, dan kiri-kanan.

$$h(x) = abs(y - y_t) + abs(x - x_t) \quad (1)$$

B. Algoritma *Flood-Fill*

Algoritma *Flood-Fill* menggunakan strategi heuristik dengan menciptakan sebuah *array* 2 dimensi yang merepresentasikan *maze*, kemudian masing-masing sel diisi nilai heuristiknya sesuai fungsi *distance* yang ditentukan. Umumnya peserta *micromouse* menggunakan algoritma ini untuk kompetisi.

C. Struktur Maze

Maze berbentuk persegi dengan ukuran 16x16. Untuk kemudahan dalam analisi pada makalah ini, akan ukuran *maze* diperkecil menjadi

5x5. Titik awal selalu dimulai dari pojok sudut *maze*. Setiap peserta harus tiba sampai bagian tengah *maze* yang berukuran 2x2, kemudian kembali ke titik awal. Jalur dalam *maze* dapat bersifat siklik, artinya bisa membentuk sirkuit. *Maze* yang diciptakan pasti mempunyai solusi sekurang-kurangnya 1 buah.

III. CARA KERJA ALGORITMA PENCARIAN JALUR

A. Algoritma A*

Nilai *upper boundary* ialah nilai *cost* maksimum jalur yang akan ditempuh robot dari titik awal menuju daerah tujuan. *Assignment* nilai *boundary* pertama bernilai kuadrat dari ukuran *maze* (robot harus mengelilingi seluruh area *maze* sebelum sampai tujuan).

Robot memeriksa daerah tetangganya, memasukkan nilai f(x) dan posisi tetangga ke dalam *priority queue*. Tetangga-tetangga tersebut akan diperiksa terlebih dahulu apakah bisa diakses langsung dari sel sekarang - dengan kata lain, tidak ada dinding penghalang. Struktur data tersebut sedemikian sehingga yang akan dikutip pertama kali dari *queue* adalah elemen yang memiliki nilai f(x) terkecil (minimum).

Kemudian *pop* elemen dan lakukan perpindahan ke posisi yang di-*pop*.

1. *Assign upper boundary* sama dengan kuadrat ukuran *maze*
2. Periksa posisi tetangga dan validasi apakah posisi tetangga *valid*. Dikatakan *valid* jika posisi tetangga masih di dalam ukuran *maze* dan dapat diakses secara langsung dari sel sekarang (tidak ada dinding penghalang.
3. Lakukan langkah 2 hingga ditemukannya sel tujuan

Untuk kembali dari bagian tengah *maze* ke titik awal, cukup menggunakan *array* simpul yang pernah dilalui, sehingga mengurangi langkah-langkah yang perlu diperiksa.

Untuk membantu dalam menyimpan struktur data, maka dipakai *Priority Queue*, dimana elemen tunggal dari *queue* ialah posisi, dan nilai g(x) dan h(x) dimana, g(x) dihitung dari jumlah langkah yang diambil dari langkah sebelumnya (bukan simpul hidup), sedangkan g(x) dihitung dari fungsi heuristik.

Selain itu, diperlukan sebuah *array* 2 dimensi yang berisi sel-sel yang sudah pernah ditempuh. Dengan adanya *array* tersebut, robot bisa bergerak ke sel yang sudah pernah dilalui tanpa melakukan pemeriksaan yang berlebih. Jika sudah mencapai *goal*, jangan melakukan pencarian lagi.

B. Algoritma *Flood-Fill*

Algoritma *Flood-fill* membutuhkan pemetaan pertama terhadap *maze* sebelum memulai perjalanan. Masing-masing sel pada *array* akan diisi dengan nilai optimistik menuju tujuan – fungsi heuristik. Tujuan robot sekarang ialah mencapai sel yang bernilai heuristik sama dengan 0.

1. Robot bergerak ke sel tetangga dengan nilai heuristik lebih kecil dari heuristik di sel yang ia berada sekarang
2. Jika robot tidak dapat menemukan sel tetangga dengan nilai heuristik yang lebih kecil, *update* nilai heuristik hingga sel-sel yang dilalui secara konsisten – yakni menurun.
3. Jika menemukan sel tetangga dengan heuristik lebih kecil, ulangi langkah 1 hingga ditemukan nilai heuristik sama dengan 0 – sudah sampai tujuan

Untuk kembali ke titik asal, lakukan hal yang sama, dengan *array* di-set ulang dan diisi nilai heuristik acuan titik akhir adalah titik dimulainya permainan.

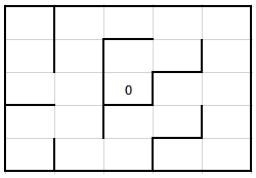
Update nilai heuristic pada sel-sel yang pernah dilalui dan tidak konsisten dapat dilakukan dengan pemanfaatan struktur data *stack* pada algoritma berikut.

- Setiap mengunjungi sebuah sel, *push* sel tersebut ke *stack*
- Perulangan selama *stack* tidak kosong:
  - *Pull* sel dari *stack.*
  - Periksa apakah heuristik sel lebih besar dari sel tetangganya.
  - Jika iya, *pull* sel lainnya dari *stack.*

- Jika tidak, *update* heuristic sel tersebut dengan heuristik terkecil sel tetangga tersebut, kemudian ditambah 1. Lalu *push* semua sel tetangganya ke dalam *stack*.

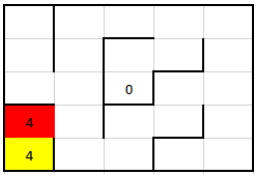
IV. PERCOBAAN

Pada percobaan ini, saya menggunakan *maze* ukuran 5x5 dengan posisi awal *mice* berada pada sudut kiri bawah.



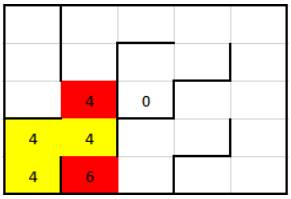
Gambar 1. Peta Maze

A. Implementasi A-*star* pada *Maze*

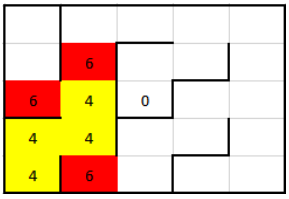


Gambar 2. Periksa tetangga, Jika hanya terdapat 1 pilihan, masuk

Periksa apakah sel tetangga dapat diakses tidak. Selain itu, perlu diperhatikan sebelum diambil sel yang akan dikunjungi terhadap nilai estimasi biaya tempuh, yaitu penjumlahan jarak tempuh sekarang dengan estimasi heuristik ke tujuan seperti Gambar 4.



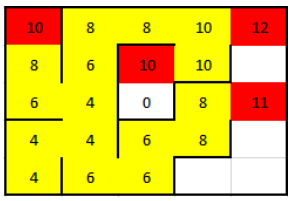
Gambar 3. Jika terdapat 2 tetangga atau lebih yang bisa dilalui, ambil f(x) minimum



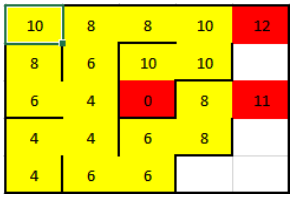
Gambar 4. Ambil sel dengan nilai f(x) minimum (warna merah)

Pemeriksaan tetap mengikuti kaidah A-*star*, sebagaimana memeriksa hingga robot *stuck*, karena secara visual benar, tetapi robot tidak bisa mengenali apa yang akan dihadapinya, baik dinding maupun penghalang. Seperti pada Gambar 6.

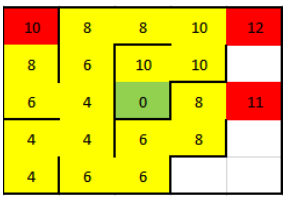
Jika robot sudah mencapai tujuan, hentikan pencarian, karena tidak perlu mencari jalan lagi.



Gambar 5 Kondisi saat dekat penghalang



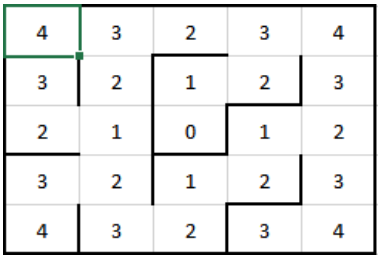
Gambar 6 Sel baris 1 kolom 1 tetap diperiksa



Gambar 7 Robot mencapai tujuan

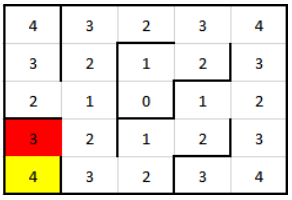
B. Implementasi *Flood-fill* pada *Maze*

Tahap pertama ialah mengisi *array representasi Maze* dengan nilai heuristik / optimistik. Pada tahap ini, dianggap tidak terdapat dinding penghalang.

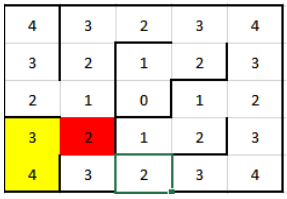


Gambar 8. Pengisian nilai heuristik

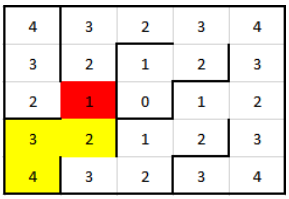
Kemudian menelusuri sel tetangga yang



Gambar 9. Langkah pemilihan sel(1)



Gambar 10. Langkah pemilihan sel(2)



Gambar 11. Langkah pemilihan sel(3)

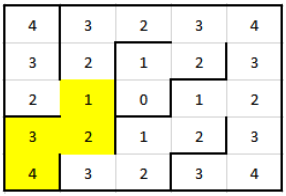
heuristiknya lebih kecil. Karena *maze* adalah area berpetak, selisih antar sel (atas-bawah, kiri-kanan) selalu bernilai 1 satuan.

Pada Gambar 11, sel selanjutnya dari sel berwarna bernilai 2, tidak bisa diambil karena sel tetangga yang diambil wajib mempunyai nilai yang lebih kecil. Untuk bisa melanjutkan perjalanan, maka sel sekarang yand didiami harus di-*update* nilainya, beserta sel-sel yang pernah dilalui

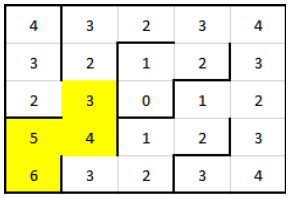
Ulangi update dan pemilihan sel seperti langkah-langkah di atas hingga mencapai tujuan.

V. PERBANDINGAN

Berdasarkan percobaan pada *maze* di atas, algoritma A* cenderung 'menelusuri' semua sel yang berada di dalam *maze* agar mendapat jalur terpendek. A* juga tidak konsisten dalam pemilihan jalur, seperti

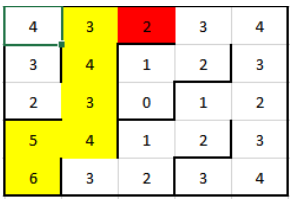


Gambar 12. Nilai sel tidak sejalan dengan sel yang akan dilalui

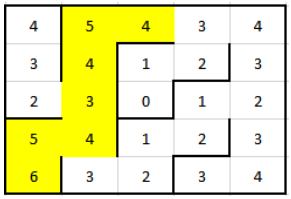


Gambar 13. Nilai heuristik setelah diperbarui

pada Gambar 5 dan 6, terdapat 3 buah sel yang belum diekspan. Sel yang akan diekspan selanjutnya ialah yang bernilai 10. Terdapat 2 buah sel yang bernilai 10 satuan, sehingga kemungkinan sel yang terpilih ialah



Gambar 14



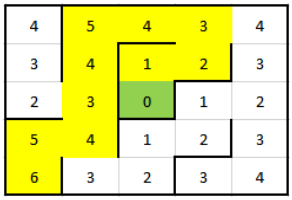
Gambar 15

sel yang berada pada posisi *row 1* dan *column 1*, Jika terjadi, maka robot harus berpindah dari sel posisi *row 2* dan *column 3* ke sel yang kembali menjauh, padahal tujuan sudah sangat dekat.

Pada kasus terburuk (*worst-case*), algoritma A* memeriksa seluruh sel di dalam *maze*.

Pada algoritma *Flood-fill*, pemeriksaan sel konsisten dengan urutan menurun, yakni memilih sel selanjutnya dengan heuristik lebih kecil. Bagaimana jika heuristic tetangga lebih besar? Untuk terus melanjutkan proses pencarian, dilakukan perbaruan nilai heuristik dari sel yang sedang ditempati. Tujuannya:

1. Melanjutkan pencarian dengan nila heuristic menurun
2. Memperbarui nilai *cost* sebenarnya jika robot melalui sel tersebut, artinya robot akan mendapatkan *cost* yang lebih besar



Gambar 16. Robot mencapai tujuan

Perhitungan ulang heuristik sel-sel yang sudah dilalui tidak perlu menggerakkan robot ke sel tersebut, karena perubahan dilakukan di dalam prosesor dan di *array* yang sudah disimpan sebelumnya. Kelebihan ini memberi nilai tambah yang signifikan untuk algoritma *flood-fill*. Selain itu, algoritma *flood-fill* juga konsisten pada jalurnya, tidak menyebar dan mengarah ke satu arah jalur.

VI. KESIMPULAN

Kedua algoritma di atas dapat digunakan pada kompetisi *micromouse*. Penekanan pada algoritma A-*star* adalah mencari jalur terpendek, bukan jalur terpendek dan tercepat. Sehingga robot akan terlihat mengeksplorasi bagian-bagian dari *maze*. Keuntungannya ialah, robot sudah mengetahui jalur mana yang akan diambil ketika kembali dari tengah *maze* ke posisi semula.

A-*star* idealnya digunakan pada *maze* yang percabangannya sedikit, sehingga simpul-simpul yang mungkin hanya sedikit atau mendekati 1, sehingga pemeriksaan simpul dilakukan hanya sekali dan tidak melompat.

Sedangkan algoritma *flood-fill* cenderung mempertahankan jalurnya agar konsisten menuju tujuan. Akan tetapi hal tersebut dibayar dengan penggunaan memori yang berlebih karena menyimpan struktur data *stack*. Selain itu, dapat terjadi perulangan dengan *worst-case*, setiap sel di-*update* ketika di-*push* dari *stack*.

Dampak tersebut dapat diperkecil dengan mengubah spesifikasi perangkat keras dari robot

tersebut, seperti prosesor yang mampu menampung memori lebih besar dan menangani proses lebih banyak pada satu satuan waktu.

Algoritma *flood-fill* idealnya digunakan ketika menghadapi *maze* dengan cabang yang banyak, karena mampu mendapatkan keputusan optimal dalam pemilihan cabang.

VII. UCAPAN TERIMA KASIH

Saya ucapkan terima kasih kepada para dosen pengajar mata kuliah strategi algoritma selama 1 semester ini. Walaupun tugas yang diberikan tidaklah sedikit, saya tetap bersyukur dan dapat memetik pelajaran dan pengalaman dari pengerjaan tugas-tugas ini. Tidak lupa saya ucapkan terima kasih kepada orang tua saya yang selalu mendengarkan keluh kesah saya selama ini.

VIII. REFERENSI

[http://robogames.net/rules/maze.php, diakses pada tanggal 24 April 2019]

[http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/A-Star-Best-FS-dan-UCS-(2018).pdf. diakses pada tanggal 25 April 2019]

[3] Ibrahim Elshamarka dan Abu Bakar Sayuti Saman, Jurnal "Design and Implementation of a Robot for Maze-Solving using Flood-Fill Algorithm," vol 56, No.5, International Journal of Computer Applications, 2012.

[4] Liu, Xiang & Gong, Daoxiong. (2011). A comparative study of A-star algorithms for search and rescue in perfect maze. 10.1109/ICEICE.2011.5777723.

[5] Barnouti, N.H., Al-Dabbagh, S.S.M. and Naser, M.A.S.(2016) "Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm," Journal of Computer and Communications, 4, 15-25. http://dx.doi.org/10.4236/jcc.2016.411002

IX. 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, 26 April 2019

Arvin Yustin
13517124