Simulasi Strategi Navigasi Layanan Robot Pengantar Pesanan di Gedung Bertingkat Menggunakan Algoritma UCS, Greedy Best-First Search dan A*

Seabastian Enrico Nathanael – 13523134
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: sebastian230405@gmail.com , 13523134@std.stei.itb.ac.id

Abstract*— Service automation in complex indoor environments such as hotels and apartments presents a significant challenge for robotic systems. One of the main tasks is efficient navigation for delivery services. This paper presents a simulation and comparative analysis of three fundamental route-finding algorithms: Uniform Cost Search (UCS), Greedy Best-First Search (Greedy BFS), and A* (A-Star), for the case of a delivery robot in a multi-story building. The environment is simulated as a weighted graph where nodes represent important locations (rooms, intersections, elevators) and edges represent paths with a specific cost that includes distance and travel time, including elevator waiting times. The simulation results show that the A* algorithm, by utilizing an admissible heuristic function, consistently produces optimal routes with much higher computational efficiency compared to UCS. Meanwhile, Greedy BFS, despite being very fast, proves to be unreliable in guaranteeing an optimal route.

Keywords*— Robotics, Robot Navigation, Pathfinding, A* Algorithm, Uniform Cost Search, Greedy Best-First Search, Multi-story Building

I. PENDAHULUAN

Perkembangan pesat dalam teknologi robotika telah membuka peluang untuk otomatisasi berbagai layanan di lingkungan manusia, mulai dari manufaktur hingga logistik. Seiring dengan kemajuan teknologi, robot layanan bergerak (*mobile service robots*) semakin banyak diadopsi untuk melakukan otomasi tugas di berbagai lingkungan dalam ruangan, seperti perkantoran, rumah sakit, dan hotel. Salah satu tugas paling umum adalah pengantaran barang, dokumen, atau makanan, yang menuntut kemampuan navigasi yang andal dan efisien.

Navigasi di dalam gedung bertingkat menghadirkan serangkaian tantangan unik yang tidak ditemukan di lingkungan datar satu lantai. Robot tidak hanya harus menavigasi koridor dan menghindari rintangan, tetapi juga harus mampu berinteraksi dengan elemen vertikal seperti lift yang menghadirkan kompleksitas vertikal. Efisiensi rute menjadi krusial, tidak hanya untuk meminimalkan waktu pengantaran tetapi juga untuk menghemat konsumsi energi baterai robot.

Pemilihan algoritma pencarian rute (*pathfinding*) yang tepat adalah kunci untuk mencapai navigasi yang otonom dan efisien. Algoritma klasik seperti Uniform Cost Search (UCS), Greedy Best-First Search (Greedy BFS), dan A* menawarkan pendekatan yang berbeda dalam menyelesaikan masalah ini. UCS menjamin rute dengan biaya terendah namun seringkali tidak efisien secara komputasi. Greedy BFS sangat cepat namun tidak menjamin optimalitas. A* mencoba menggabungkan keunggulan keduanya.

Makalah ini bertujuan untuk melakukan analisis perbandingan kinerja ketiga algoritma tersebut melalui sebuah simulasi. Kami memodelkan sebuah lingkungan gedung bertingkat generik dan mensimulasikan bagaimana setiap algoritma mencari rute untuk sebuah tugas pengantaran. Kinerja diukur berdasarkan tiga metrik utama: optimalitas rute (total biaya), efisiensi pencarian (jumlah simpul yang dieksplorasi), dan waktu komputasi.

II. DASAR TEORI

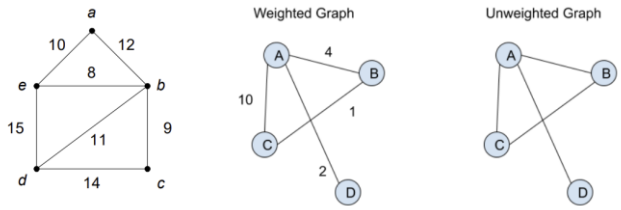
A. Graf Berbobot

Graf merupakan representasi matematis yang terdiri atas simpul (node/vertex) dan sisi (edge) yang menghubungkan antar simpul. Dalam konteks pemodelan lingkungan fisik seperti gedung bertingkat, graf yang digunakan bersifat *berbobot* (weighted graph), yaitu setiap sisi memiliki nilai bobot tertentu. Graf berbobot dapat didefinisikan sebagai

$$G = (V, E)$$

di mana:

- V adalah himpunan simpul yang merepresentasikan lokasi-lokasi penting dalam gedung,
- E adalah himpunan sisi yang menghubungkan pasangan simpul dalam V, dan
- setiap sisi $(u, v) \in E$ memiliki bobot $w(u, v)$ yang mencerminkan biaya perpindahan antar lokasi, seperti jarak, waktu tempuh, atau energi yang dibutuhkan.



Gambar 1. Contoh Graf Berbobot

Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf.

B. *Uniform Cost Search (UCS)*

Uniform Cost Search adalah algoritma pencarian yang secara sistematis mengeksplorasi graf untuk menemukan rute dengan total biaya terendah dari simpul awal ke semua simpul lainnya. UCS merupakan generalisasi dari Algoritma Dijkstra dan bekerja dengan menggunakan struktur data antrian prioritas (*priority queue*) untuk selalu memperluas simpul n yang memiliki total biaya $g(n)$ (jarak dari simpul awal) paling rendah[1]. Karena UCS tidak menggunakan informasi heuristik mengenai lokasi tujuan, pencariannya menyebar ke segala arah dari titik awal, mirip seperti gelombang. Meskipun proses ini menjamin penemuan rute yang paling optimal dari segi biaya, ia cenderung tidak efisien secara komputasi pada ruang pencarian yang besar karena akan memeriksa banyak jalur yang tidak relevan[2].

C. *Greedy Best-First Search*

Greedy Best-First Search (Greedy BFS) adalah algoritma pencarian terinformasi (*informed search*) yang menggunakan fungsi heuristik $h(n)$ untuk memandu pencariannya. Fungsi ini memberikan "estimasi" atau "tebakan" mengenai seberapa dekat simpul n dengan simpul tujuan. Algoritma ini disebut "rakus" (*greedy*) karena di setiap langkah, ia selalu memilih untuk mengeksplorasi simpul yang memiliki nilai $h(n)$ terendah, dengan harapan dapat mencapai tujuan secepat mungkin [3]. Pendekatan ini seringkali sangat cepat dalam menemukan sebuah solusi. Namun, karena ia sepenuhnya mengabaikan biaya $g(n)$ yang telah ditempuh, Greedy BFS tidak memiliki jaminan optimalitas. Ia bisa dengan mudah tertipu oleh jalur yang terlihat pendek secara heuristik namun sebenarnya memiliki biaya total yang sangat tinggi, atau bahkan terjebak dalam jalan buntu [3].

D. *A* (A-Star)*

Algoritma $A*$ dapat dianggap sebagai mahakarya yang menggabungkan keandalan UCS dan efisiensi Greedy BFS. $A*$ adalah algoritma pencarian terinformasi yang mengevaluasi setiap simpul n menggunakan fungsi evaluasi $f(n)$ yang ikonik:

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

Di sini, $g(n)$ adalah komponen biaya aktual dari UCS yang memastikan pencarian memperhitungkan biaya yang sudah dikeluarkan, sementara $h(n)$ adalah komponen heuristik dari Greedy BFS yang mengarahkan pencarian secara efisien ke tujuan. Kombinasi ini memungkinkan A* untuk melakukan pencarian yang terarah tanpa mengorbankan jaminan optimalitas. Kunci dari A* adalah penggunaan fungsi heuristik yang *admisibel*, yaitu $h(n)$ yang tidak pernah melebih-lebihkan biaya sebenarnya untuk mencapai tujuan. Jika syarat ini terpenuhi, A* dijamin akan menemukan rute dengan biaya terendah, seringkali dengan kecepatan yang jauh melampaui UCS [4].

III. Metodologi Penelitian dan Pembahasan

A. *Pemodelan Lingkungan*

Sebuah lingkungan gedung virtual dibuat secara prosedural untuk menciptakan ruang lingkup pengujian yang besar dan kompleks. Spesifikasi gedung adalah sebagai berikut:

1. Dimensi: 25 lantai, dengan 50 ruangan dan 10 koridor setiap lantai.
2. Struktur Lift: Terdapat dua poros lift dengan karakteristik berbeda untuk menciptakan skenario pengujian yang menantang:
   a. Lift A (Utama): Lift penumpang normal yang berlokasi di dekat awal koridor ($x = 0$).
   b. Lift B (Barang/Jebakan): Lift layanan yang berlokasi di ujung koridor ($x = 110$), secara posisi lebih dekat ke beberapa ruangan tujuan, namun sengaja dibuat sangat lambat.

B. *Model Biaya (Cost)*

Biaya setiap sisi (edge) dalam graf merepresentasikan estimasi waktu tempuh dalam satuan detik:

1. Biaya Horizontal: Biaya bergerak dari ruangan ke koridor atau antar koridor ditetapkan dengan nilai rendah (misalnya, 5 dan 10 detik).
2. Biaya Vertikal (Lift):
   a. Lift A: Biaya dihitung sebagai $wA = 30 + 5 \times \Delta L$, di mana 30 detik adalah waktu tunggu dan 5 detik adalah waktu tempuh per lantai (ΔL).
   b. Lift B: Biaya dihitung sebagai $wB = 150 + 10 \times \Delta L$, di mana 150 detik adalah waktu tunggu yang sangat lama dan 10 detik adalah waktu tempuh per lantai.

C. *Proses Permodelan Graf*

Lingkungan gedung virtual tidak dibuat secara manual, melainkan dibangkitkan (dihasilkan) secara prosedural oleh sebuah fungsi dalam program Python. Pendekatan ini memastikan bahwa lingkungan pengujian bersifat konsisten, kompleks, dan dapat dengan mudah diskalakan. Proses pembentukan graf ini mengikuti langkah-langkah logis sebagai berikut:

1. Inisialisasi Simpul (Node) Struktural per Lantai

Proses dimulai dengan melakukan iterasi untuk setiap lantai dari 1 hingga 25. Pada setiap lantai, simpul-simpul utama yang

menjadi "tulang punggung" lantai tersebut dibuat terlebih dahulu:

1. Dua Simpul Lift: Dua jenis simpul lift dibuat pada setiap lantai untuk merepresentasikan dua poros lift yang berbeda.
    a. $L\{f\} - LIFTA$: Merepresentasikan "Lift Utama" yang cepat. Ditempatkan pada koordinat strategis awal (misalnya, $x = 0$).
    b. $L\{f\} - LIFTB$: Merepresentasikan "Lift Barang" atau "Jebakan" yang lambat. Ditempatkan pada koordinat ujung koridor (misalnya, $x = 110$) untuk membuatnya terlihat menarik dari beberapa lokasi.
2. Sepuluh Simpul Koridor: Sebanyak sepuluh simpul koridor ($L\{f\} - C1$ hingga $L\{f\} - C10$) dibuat dan diletakkan secara berurutan di sepanjang sumbu-x, membentuk sebuah lorong utama virtual.

2. Pembangunan Konektivitas Horizontal per Lantai

Setelah semua simpul struktural ada, sisi (*edge*) ditambahkan untuk menghubungkan mereka secara horizontal pada lantai yang sama:

1. Koneksi Antar Koridor: Simpul koridor dihubungkan satu sama lain secara sekuensial (C1 ke C2, C2 ke C3, dan seterusnya) dengan biaya tetap (misalnya, 10 detik), menciptakan jalur utama yang dapat dilalui.
2. Koneksi Koridor ke Lift: Titik akses ke lift didefinisikan dengan menghubungkan koridor ujung ke masing-masing lift. Koridor pertama (C1) dihubungkan ke LIFTA, dan koridor terakhir (C10) dihubungkan ke LIFTB.
3. Koneksi Ruangan: Untuk setiap lantai, 50 simpul ruangan ($L\{f\} - R1$ hingga $L\{f\} - R50$) dibuat. Setiap ruangan kemudian dihubungkan ke segmen koridor terdekatnya (dengan asumsi 5 ruangan per segmen koridor) dengan biaya rendah (misalnya, 5 detik).

3. Pembangunan Konektivitas Vertikal Antar Lantai

Langkah terakhir adalah menghubungkan semua lantai secara vertikal dengan menciptakan "poros lift":

1. Poros Lift A (Cepat): Program melakukan iterasi untuk menghubungkan setiap simpul LIFTA di satu lantai dengan semua simpul LIFTA di lantai lainnya. Setiap sisi vertikal ini diberi biaya yang relatif rendah (lift_cost_a = 30 + abs(f1 - f2) * 5).
2. Poros Lift B (Lambat/Jebakan): Proses yang sama dilakukan untuk simpul LIFTB. Namun, setiap sisi vertikalnya diberi biaya yang jauh lebih tinggi (lift_cost_b = 150 + abs(f1 - f2) * 10). Langkah inilah yang secara efektif menciptakan "jebakan" di dalam graf.

Melalui proses tiga langkah ini, sebuah graf besar yang terdiri dari 1550 simpul berhasil dibuat secara dinamis. Hasilnya adalah sebuah lingkungan virtual yang kompleks dan menantang, menyediakan sebuah testbed yang adil dan realistis untuk menguji dan membandingkan keunggulan serta kelemahan dari setiap algoritma pencarian rute.

D. Aplikasi Algoritma UCS

Pseudocode untuk algoritma UCS :

```
function UniformCostSearch(graph, start, goal)
  // Inisialisasi antrian prioritas dengan node awal (prioritas 0)
  open_list = new PriorityQueue()
  open_list.add(start, 0)

  // Lacak biaya terendah untuk mencapai setiap node
  cost_so_far = new Map()
  cost_so_far[start] = 0

  while open_list is not empty
    current = open_list.pop_lowest_cost()

    if current == goal
      break // Tujuan ditemukan

    for each neighbor of current in graph
      new_cost = cost_so_far[current] + graph.cost(current, neighbor)
      if neighbor not in cost_so_far or new_cost < cost_so_far[neighbor]
        cost_so_far[neighbor] = new_cost
        priority = new_cost // Prioritas hanya berdasarkan biaya
        open_list.add(neighbor, priority)

  return path, cost_so_far[goal]
```

Proses operasional UCS dimulai dengan inisialisasi, di mana simpul awal *(start node*) dimasukkan ke dalam sebuah antrian prioritas yang menggunakan total biaya kumulatif ($g(n)$) sebagai dasar prioritas, dengan nilai awal $g(start\_node)$ adalah nol. Dalam setiap iterasi, UCS akan mengekstraksi simpul dengan nilai $g(n)$ terendah dari antrian, yang kemudian diekspansi dan dianggap telah dikunjungi. Selanjutnya, algoritma mengevaluasi semua simpul tetangga yang terhubung dengan simpul tersebut. Untuk setiap tetangga, dihitung *new_cost* sebagai penjumlahan antara $g(n)$ dari simpul saat ini dan bobot sisi yang menghubungkannya dengan tetangga tersebut. Jika *new_cost* lebih kecil dari biaya yang tercatat sebelumnya atau jika tetangga tersebut belum pernah dikunjungi, maka nilai biaya untuk simpul tersebut diperbarui dalam antrian prioritas. Proses ini terus berulang hingga simpul yang diekstraksi dari antrian adalah simpul tujuan (goal node), dan pada titik tersebut algoritma berhenti dengan hasil rute terpendek secara biaya yang optimal.

E. Aplikasi Algoritma Greedy Best-First Search

Pseudocode untuk algoritma Greedy Best-First Search :

```
function GreedyBestFirstSearch(graph, start, goal)
  // Inisialisasi antrian prioritas dengan node awal
  open_list = new PriorityQueue()
  // Prioritas hanya berdasarkan nilai heuristik
  open_list.add(start, heuristic(start, goal))

  visited = new Set()

  while open_list is not empty
    current = open_list.pop_lowest_heuristic()

    if current == goal
      break // Tujuan ditemukan

    visited.add(current)

    for each neighbor of current in graph
      if neighbor not in visited
        priority = heuristic(neighbor, goal)
        open_list.add(neighbor, priority)

  return path
```

Algoritma ini memulai proses pencarian dengan inisialisasi simpul awal ke dalam antrian prioritas, di mana setiap simpul diberi prioritas berdasarkan nilai heuristik $h(n)$ yang merupakan estimasi biaya dari simpul tersebut ke simpul tujuan. Pada setiap iterasi, algoritma secara "rakus" memilih simpul dengan nilai heuristik terkecil, yaitu yang dianggap paling dekat ke tujuan berdasarkan estimasi. Setelah simpul diekspansi, algoritma mengevaluasi semua simpul tetangga yang terhubung dengannya. Untuk setiap tetangga yang belum dikunjungi, nilai $h(neighbor)$ dihitung dan dimasukkan ke dalam antrian prioritas tanpa memperhitungkan biaya aktual yang telah dikeluarkan ($g(n)$). Proses ini diulang hingga simpul yang diekstraksi dari antrian adalah simpul tujuan. Karena mengabaikan biaya nyata dari simpul awal, Greedy BFS dapat menyelesaikan pencarian dengan cepat, tetapi tidak menjamin solusi yang ditemukan adalah solusi optimal.

F. Aplikasi Algoritma A (A-Star)*

Pseudocode untuk algoritma A* :

```
function AStarSearch(graph, start, goal)
  // Inisialisasi antrian prioritas dengan node awal
  open_list = new PriorityQueue()
  // Prioritas berdasarkan f(n) = g(n) + h(n)
  open_list.add(start, 0 + heuristic(start, goal))

  // Lacak biaya terendah (g score) untuk mencapai setiap node
  g_score = new Map()
  g_score[start] = 0

  while open_list is not empty
    current = open_list.pop_lowest_f_score()

    if current == goal
      break // Tujuan ditemukan

    for each neighbor of current in graph
      // Hitung g score tentatif untuk tetangga
      tentative_g_score = g_score[current] + graph.cost(current, neighbor)

      // Jika jalur baru ini lebih baik dari yang pernah ada
      if tentative_g_score < g_score.get(neighbor, infinity)
        g_score[neighbor] = tentative_g_score
        // Hitung f score baru dan masukkan ke antrian
        f_score = tentative_g_score + heuristic(neighbor, goal)
        open_list.add(neighbor, f_score)

  return path, g_score[goal]
```

Proses pencarian A* dimulai dengan menyimpan simpul awal dalam antrian prioritas, dengan $g(start\_node)$ diinisialisasi sebagai nol. Setiap iterasi mengekstraksi simpul dengan nilai $f(n)$ terendah sebagai simpul yang paling menjanjikan untuk diekspansi. Untuk setiap simpul tetangga yang dievaluasi, algoritma menghitung nilai tentative g_score, yaitu biaya dari awal ke tetangga melalui simpul saat ini. Jika nilai ini lebih rendah dari $g(n)$ yang telah tercatat sebelumnya, maka catatan tersebut diperbarui dan nilai $f(n)$ yang baru dihitung serta dimasukkan kembali ke antrian prioritas. Proses berlanjut hingga simpul yang diekstraksi dari antrian adalah simpul tujuan.

G. Pemrosesan Rute dan Pengukuran

Setelah model graf lingkungan gedung berhasil dibuat secara prosedural, langkah selanjutnya adalah menjalankan proses pencarian rute untuk setiap algoritma secara sistematis. Proses ini diatur oleh sebuah kode utama (main.py) yang berfungsi sebagai orkestrator eksperimen untuk memastikan perbandingan yang adil dan konsisten.

Prosedur pemrosesan rute untuk setiap uji coba mengikuti langkah-langkah berikut:

1. Penentuan Titik Awal dan Tujuan: Sebelum simulasi dimulai, satu simpul (node) di dalam graf ditetapkan sebagai titik awal (start_node) dan satu simpul lain sebagai titik tujuan (goal_node). Dalam implementasi ini, titik-titik tersebut dapat diterima melalui input pengguna

dinamis, memungkinkan pengujian berbagai skenario rute dengan mudah.

2. Eksekusi Algoritma secara Berurutan: Skrip utama akan menjalankan ketiga algoritma Uniform Cost Search (UCS), Greedy Best-First Search (Greedy BFS), dan A* secara bergantian. Poin krusial di sini adalah setiap algoritma diuji pada instansi graf yang identik dan dengan pasangan titik awal-tujuan yang sama. Hal ini menghilangkan variabel luar dan memastikan bahwa perbedaan hasil murni disebabkan oleh karakteristik internal setiap algoritma.
3. Pencatatan dan Pengukuran Kinerja: Selama dan setelah eksekusi setiap algoritma, program secara otomatis mencatat tiga metrik kinerja utama. Metrik-metrik ini dipilih untuk memberikan analisis yang multi-dimensi, mencakup kualitas solusi dan efisiensi proses.

Metrik Kinerja yang Diukur:

1. Biaya Rute (detik): Metrik ini mengukur kualitas solusi yang ditemukan. Nilainya adalah total akumulasi dari semua bobot (biaya) pada sisi (edge) di sepanjang rute dari awal hingga tujuan. Dalam konteks simulasi ini, biaya diinterpretasikan sebagai waktu tempuh virtual robot dalam detik. Nilai yang lebih rendah menunjukkan rute yang lebih baik, lebih cepat, dan lebih efisien.
2. Simpul Dieksplorasi: Metrik ini mengukur efisiensi algoritmik. Nilainya adalah jumlah total simpul unik yang diperiksa atau diekspansi oleh algoritma selama proses pencarian. Jumlah simpul yang lebih sedikit menandakan bahwa algoritma lebih "cerdas" dan terarah, tidak membuang sumber daya komputasi untuk memeriksa jalur-jalur yang tidak relevan.
3. Waktu Komputasi (ms): Metrik ini mengukur efisiensi praktis atau kinerja dunia nyata dari algoritma pada perangkat keras yang digunakan. Nilainya adalah waktu jam (wall-clock time) dalam milidetik yang dibutuhkan oleh prosesor untuk menyelesaikan eksekusi algoritma. Metrik ini sangat dipengaruhi oleh jumlah simpul yang dieksplorasi dan menjadi indikator penting untuk kelayakan algoritma pada aplikasi *real-time*.

Dengan mengumpulkan dan membandingkan ketiga metrik ini di berbagai skenario uji coba, analisis yang komprehensif mengenai kekuatan dan kelemahan masing-masing strategi navigasi dapat dilakukan.

H. Simulasi dan Analisis Hasil

Untuk menganalisis kinerja ketiga algoritma, beberapa skenario simulasi dijalankan dengan titik awal dan tujuan yang berbeda. Analisis berikut didasarkan pada hasil yang didapat dari eksekusi program.

1. Analisis Skenario Jebakan Navigasi (L1-R1 ke L25-R50)

```
Membuat model gedung besar dengan 2 jenis lift...
Menghubungkan lift antar lantai...
Model gedung selesai dibuat dengan 1550 node.

--- Masukkan Titik Awal ---
Masukkan Lantai Awal (1-25): 1
Masukkan Nomor Kamar Awal (1-50): 1

--- Masukkan Titik Tujuan ---
Masukkan Lantai Tujuan (1-25): 25
Masukkan Nomor Kamar Tujuan (1-50): 50

==================================================
      MEMULAI SIMULASI PENCARIAN RUTE
==================================================
Mencari rute dari L1-R1 ke L25-R50...

[1] Menjalankan Uniform Cost Search...
UCS selesai.
[2] Menjalankan Greedy Best-First Search...
Greedy BFS selesai.
[3] Menjalankan A* Search...
A* selesai.

============================================================
                HASIL AKHIR PERBANDINGAN
============================================================
Metrik                  | UCS           | Greedy BFS      | A*
------------------------------------------------------------------------------
Biaya Rute (detik)      | 270.00        | 510.00          | 270.00
Simpul Dieksplorasi     | 1549          | 49              | 109
Waktu Komputasi (ms)    | 1.9740        | 0.1593          | 0.4923

============================================================
```

Gambar 2. Hasil Percobaan simulasi 1
Sumber : Dokumen Penulis

Skenario ini adalah pengujian paling krusial, dirancang untuk memaksa algoritma memilih antara jalur yang terlihat dekat (via Lift B) dan jalur yang sebenarnya efisien (via Lift A). Hasil dari simulasi ini disajikan pada Gambar 2.

Dari hasil ini, terlihat perbedaan kinerja yang sangat signifikan:

1. Greedy BFS Gagal. Algoritma ini sepenuhnya tertipu oleh posisi Lift B yang secara heuristik lebih dekat ke tujuan L25-R50. Tanpa mempertimbangkan biaya waktu tunggu yang sangat tinggi, ia memilih rute melalui Lift B. Akibatnya, ia menghasilkan rute dengan Biaya Rute sebesar 510.0 detik. Ini adalah solusi yang sangat tidak efisien, hampir dua kali lipat (88% lebih mahal) dari biaya rute optimal. Meskipun proses pencariannya sangat cepat (hanya 49 simpul), solusi yang dihasilkannya tidak dapat diterima.
2. *UCS dan A** Berhasil dengan Optimal. Kedua algoritma ini terbukti "pintar" dan tidak terjebak. Mereka mampu menghitung bahwa total biaya melalui Lift B terlalu mahal. Keduanya berhasil mengidentifikasi rute yang benar melalui Lift A yang lebih cepat, dan mencapai solusi optimal dengan Biaya Rute 270.0 detik.
3. *A** Terbukti Unggul. Di antara dua algoritma optimal, A* menunjukkan efisiensi yang luar biasa. Ia menemukan rute terbaik dengan hanya mengeksplorasi 109 simpul. Sebagai perbandingan, UCS harus memeriksa 1549 simpul untuk menjamin hasil yang sama. Ini membuktikan A* mampu mencapai optimalitas tanpa mengorbankan efisiensi komputasi

1. Analisis Biaya Rute

Terdapat perbedaan Biaya Rute yang sangat drastis. UCS dan A* menemukan rute optimal dengan biaya 270.00, sementara Greedy BFS menghasilkan rute dengan biaya 510.00. Hasil ini dengan jelas menunjukkan kegagalan algoritma Greedy BFS. Greedy BFS tertipu oleh fungsi

heuristik. Karena tujuan (L25-R50) secara posisi dekat dengan Lift B yang lambat, heuristik memberikan sinyal bahwa jalur melalui Lift B adalah yang paling menjanjikan. Tanpa mempertimbangkan biaya aktual ($g(n)$) yang sangat tinggi dari Lift B, Greedy secara "rakus" memilih jalur tersebut. Akibatnya, solusi yang dihasilkannya 88% lebih mahal daripada rute optimal. Sebaliknya, A* berhasil menghindari jebakan karena fungsi evaluasinya $f(n) = g(n) + h(n)$ memperhitungkan biaya tinggi dari Lift B, sehingga ia dengan cerdas memilih rute melalui Lift A yang lebih efisien secara keseluruhan.

2. Analisis Efisiensi Pencarian
   a. Uniform Cost Search (UCS): UCS menjadi tolok ukur untuk biaya optimal, namun dengan efisiensi terendah. Ia harus memeriksa 1549 simpul dan memakan waktu 1.9740 ms, menunjukkan sifatnya yang tidak efisien untuk skala besar.
   b. Greedy Best-First Search (Greedy BFS): Meskipun menghasilkan solusi yang buruk, proses pencariannya sangat cepat (hanya 49 simpul dan 0.1593 ms). Ini menunjukkan bahaya dari algoritma yang mengorbankan keandalan demi kecepatan pencarian.
   c. A* (A-Star): A* menunjukkan keseimbangan yang sempurna. Ia menemukan rute optimal dengan hanya memeriksa 109 simpul, lebih dari 14 kali lebih efisien daripada UCS. Waktu komputasinya (0.4923 ms) juga sangat cepat, membuktikan kemampuannya untuk mencapai optimalitas tanpa mengorbankan efisiensi.

Kesimpulan untuk Skenario ini adalah skenario ini secara definitif membuktikan bahwa Greedy BFS tidak dapat diandalkan untuk lingkungan navigasi yang kompleks. Ia sangat rentan terhadap heuristik yang menyesatkan. Sebaliknya, A* menunjukkan keunggulannya yang superior sebagai algoritma yang robust, cerdas, dan efisien, menjadikannya pilihan ideal untuk aplikasi robotik dunia nyata.

2. Analisis Skenario Rute Non-Jebakan (L25-R50 ke L1-R1)

```
Membuat model gedung besar dengan 2 jenis lift...
Menghubungkan lift antar lantai...
Model gedung selesai dibuat dengan 1550 node.

--- Masukkan Titik Awal ---
Masukkan Lantai Awal (1-25): 25
Masukkan Nomor Kamar Awal (1-50): 50

--- Masukkan Titik Tujuan ---
Masukkan Lantai Tujuan (1-25): 1
Masukkan Nomor Kamar Tujuan (1-50): 1

================================================
      MEMULAI SIMULASI PENCARIAN RUTE
================================================
Mencari rute dari L25-R50 ke L1-R1...

[1] Menjalankan Uniform Cost Search...
UCS selesai.
[2] Menjalankan Greedy Best-First Search...
Greedy BFS selesai.
[3] Menjalankan A* Search...
A* selesai.

============================================================
               HASIL AKHIR PERBANDINGAN
============================================================
Metrik                  | UCS            | Greedy BFS     | A*
----------------------------------------------------------------------------
Biaya Rute (detik)      | 270.00         | 270.00         | 270.00
Simpul Dieksplorasi     | 965            | 19             | 55
Waktu Komputasi (ms)    | 1.1176         | 0.1308         | 0.4853

============================================================
```

Gambar 3. Hasil Percobaan Simulasi 2
Sumber : Dokumen Penulis

Skenario pengujian ini mensimulasikan perjalanan robot dari lokasi tengah gedung (Lantai 25, Kamar 50) menuju titik awal gedung (Lantai 1, Kamar 1). Skenario ini berfungsi sebagai pembanding, di mana arah perjalanan dibalik. Titik tujuan (L1-R1) berada dekat dengan jalur optimal (Lift A), sehingga tidak ada jebakan yang aktif untuk algoritma Greedy.

1. Analisis Biaya Rute

Ketiga algoritma berhasil menemukan rute dengan Biaya Rute yang identik dan optimal, yaitu 270.00 detik. Dalam kasus ini, jebakan tidak berhasil memicu Greedy BFS. Fungsi heuristik dengan benar mengidentifikasi bahwa jalur melalui Lift A secara posisi jauh lebih menarik untuk mencapai tujuan L1-R1. Karena "petunjuk" dari heuristik akurat, strategi "rakus" dari Greedy BFS secara kebetulan menuntunnya ke jalur yang memang paling efisien. Ini menunjukkan skenario "kasus terbaik" untuk algoritma Greedy.

2. Analisis Efisiensi Pencarian
   a. Uniform Cost Search (UCS): UCS tetap menjadi yang paling tidak efisien dengan 965 simpul dieksplorasi dan waktu 1.1176 ms.
   b. Greedy Best-First Search (Greedy BFS): Menjadi yang paling efisien dalam skenario ini. Dengan panduan heuristik yang sangat akurat, ia hanya perlu memeriksa 19 simpul dan selesai dalam 0.1308 ms.
   c. A* (A-Star): Menunjukkan kinerja yang sangat baik. Dengan hanya 55 simpul dieksplorasi dan waktu 0.4853 ms, ia tetap sangat efisien sambil membawa jaminan optimalitas yang tidak dimiliki oleh Greedy.

Kesimpulan untuk Skenario ini adalah skenario dari L25-R50 ke L1-R1 ini mengkonfirmasi bahwa ketika jalur optimal juga merupakan jalur yang terlihat paling jelas secara heuristik, semua algoritma dapat menemukan solusi terbaik. Pembeda

utamanya adalah efisiensi, di mana algoritma yang memanfaatkan heuristik (Greedy BFS dan A*) secara dramatis mengungguli algoritma UCS. Skenario ini penting untuk menunjukkan bahwa A* tidak hanya andal dalam skenario sulit, tetapi juga tetap sangat efisien dalam skenario yang lebih mudah.

3. Analisis Skenario Rute Alternatif (L12-R25 ke L1-R1)

```
Membuat model gedung besar dengan 2 jenis lift...
Menghubungkan lift antar lantai...
Model gedung selesai dibuat dengan 1550 node.

--- Masukkan Titik Awal ---
Masukkan Lantai Awal (1-25): 12
Masukkan Nomor Kamar Awal (1-50): 25

--- Masukkan Titik Tujuan ---
Masukkan Lantai Tujuan (1-25): 1
Masukkan Nomor Kamar Tujuan (1-50): 1

================================================
      MEMULAI SIMULASI PENCARIAN RUTE
================================================
Mencari rute dari L12-R25 ke L1-R1...

[1] Menjalankan Uniform Cost Search...
UCS selesai.
[2] Menjalankan Greedy Best-First Search...
Greedy BFS selesai.
[3] Menjalankan A* Search...
A* selesai.

============================================================
               HASIL AKHIR PERBANDINGAN
============================================================
Metrik                  | UCS            | Greedy BFS     | A*
------------------------------------------------------------------------------
Biaya Rute (detik)      | 155.00         | 155.00         | 155.00
Simpul Dieksplorasi     | 459            | 14             | 31
Waktu Komputasi (ms)    | 0.5735         | 0.0807         | 0.4811

============================================================
```

Gambar 3. Hasil Percobaan Simulasi 3
Sumber : Dokumen Penulis

Skenario pengujian ini mensimulasikan perjalanan robot dari lokasi tengah gedung (Lantai 12, Kamar 25) menuju titik awal gedung (Lantai 1, Kamar 1). Analisis hasil ini memberikan wawasan penting tentang bagaimana setiap algoritma berperilaku dalam skenario non-jebakan yang lebih umum.

1. Analisis Biaya Rute

Ketiga algoritma UCS, Greedy BFS, dan A* semuanya berhasil menemukan rute dengan Biaya Rute yang identik dan optimal, yaitu 155.00. Hasil ini menunjukkan bahwa untuk skenario ini, "jebakan" (Lift B yang lambat) tidak berhasil menipu algoritma Greedy. Alasannya adalah karena titik tujuan (L1-R1) berlokasi sangat dekat dengan Lift A (lift optimal). Fungsi heuristik, yang menghitung estimasi jarak ke tujuan, dengan benar mengidentifikasi bahwa jalur melalui Lift A secara posisi jauh lebih menarik daripada jalur melalui Lift B yang berada di ujung lain gedung.

2. Analisis Efisiensi Pencarian

a. Uniform Cost Search (UCS): UCS menjadi algoritma yang paling tidak efisien. Untuk menemukan rute optimal, ia harus memeriksa 459 simpul. Ini disebabkan oleh sifatnya yang "buta", di mana ia mengeksplorasi semua kemungkinan jalur berbiaya rendah secara merata tanpa arahan menuju tujuan. Akibatnya, waktu komputasinya menjadi yang paling tinggi di antara ketiganya, yaitu 0.5735 ms.
b. Greedy Best-First Search (Greedy BFS): Dalam skenario ini, Greedy BFS menunjukkan efisiensi pencarian yang baik. Dengan hanya fokus pada heuristik, ia mampu sampai ke tujuan dengan hanya memeriksa 14 simpul. Ini menjadikannya algoritma dengan waktu komputasi tercepat, hanya 0.0807 ms.
c. *A (A-Star):* A* menunjukkan perannya sebagai penyeimbang yang sempurna. Ia jauh lebih efisien daripada UCS, dengan hanya memeriksa 31 simpul, kurang dari 7% dari total eksplorasi UCS. Meskipun tidak secepat Greedy, ia tetap sangat efisien. A* tidak hanya mengikuti heuristik, tetapi juga mempertimbangkan biaya nyata $(g(n))$, membuatnya sedikit lebih berhati-hati namun tetap sangat terarah. Waktu komputasinya (0.4811 ms) mencerminkan keseimbangan antara kecepatan dan metodologi yang cermat ini.

Kesimpulan untuk Skenario ini adalah skenario dari L12-R25 ke L1-R1 merupakan contoh dari kasus di mana heuristik sederhana sangat efektif. Hasil ini mengkonfirmasi bahwa ketika jalur optimal juga merupakan jalur yang terlihat paling jelas, semua algoritma dapat menemukan solusi terbaik. Dalam kondisi seperti ini, pembanding utama adalah efisiensi. Algoritma yang memanfaatkan heuristik (Greedy BFS dan A*) mengungguli algoritma buta (UCS). A* membuktikan nilainya sebagai algoritma yang robust, ia efisien seperti Greedy namun tetap membawa jaminan optimalitas yang tidak dimiliki oleh Greedy, menjadikannya pilihan yang aman dan cerdas untuk berbagai kondisi.

4. Analisis Skenario Rute Alternatif (L1-R50 ke L25-R1)

```
Membuat model gedung besar dengan 2 jenis lift...
Menghubungkan lift antar lantai...
Model gedung selesai dibuat dengan 1550 node.

--- Masukkan Titik Awal ---
Masukkan Lantai Awal (1-25): 1
Masukkan Nomor Kamar Awal (1-50): 50

--- Masukkan Titik Tujuan ---
Masukkan Lantai Tujuan (1-25): 25
Masukkan Nomor Kamar Tujuan (1-50): 1

================================================
      MEMULAI SIMULASI PENCARIAN RUTE
================================================
Mencari rute dari L1-R50 ke L25-R1...

[1] Menjalankan Uniform Cost Search...
UCS selesai.
[2] Menjalankan Greedy Best-First Search...
Greedy BFS selesai.
[3] Menjalankan A* Search...
A* selesai.

============================================================
               HASIL AKHIR PERBANDINGAN
============================================================
Metrik                  | UCS            | Greedy BFS     | A*
------------------------------------------------------------------------------
Biaya Rute (detik)      | 270.00         | 270.00         | 270.00
Simpul Dieksplorasi     | 1013           | 19             | 55
Waktu Komputasi (ms)    | 1.1884         | 0.1346         | 0.5489

============================================================
```

Gambar 4. Hasil Percobaan Simulasi 4
Sumber : Dokumen Penulis

Skenario pengujian ini dirancang untuk melihat bagaimana algoritma bereaksi ketika memulai dari satu ujung gedung (dekat Lift B/jebakan) dan bertujuan ke ujung lainnya (dekat Lift A/optimal).

1. Analisis Biaya Rute

Seperti pada beberapa kasus sebelumnya, ketiga algoritma berhasil menemukan rute dengan biaya optimal yang sama, yaitu 270.00 detik. Hasil ini menunjukkan bahwa jebakan kembali tidak berhasil pada algoritma Greedy. Meskipun titik awal (L1-R50) berada sangat dekat dengan Lift B yang lambat, titik tujuan (L25-R1) berada dekat dengan Lift A. Fungsi heuristik, yang menjadi satu-satunya panduan bagi Greedy BFS, dengan cerdas menghitung bahwa tujuan akhir secara posisi jauh lebih dekat ke Lift A. Akibatnya, algoritma Greedy tidak tergoda untuk menggunakan Lift B di dekatnya. Sebaliknya, ia membuat keputusan yang benar untuk menempuh perjalanan sepanjang koridor di Lantai 1 untuk mencapai Lift A, yang merupakan jalur optimal.

2. Analisis Efisiensi Pencarian

a. Uniform Cost Search (UCS): UCS tetap menjadi yang paling tidak efisien, membutuhkan eksplorasi 1013 simpul untuk menjamin hasil optimalnya. Waktu komputasinya adalah yang terlama, yaitu 1.1884 ms.

b. Greedy Best-First Search (Greedy BFS): Menjadi yang paling efisien dalam skenario ini. Dengan panduan heuristik yang akurat, ia hanya perlu memeriksa 19 simpul dan selesai dalam 0.1346 ms.

c. *A (A-Star):** Menempati posisi ideal di antara keduanya. Dengan 55 simpul yang dieksplorasi dan waktu komputasi 0.5489 ms, A* jauh lebih efisien daripada UCS namun tetap membawa jaminan optimalitas yang tidak dimiliki oleh Greedy.

Kesimpulan untuk Skenario ini adalah skenario dari L1-R50 ke L25-R1 ini memberikan pemahaman yang lebih dalam mengenai sebuah "jebakan" yang hanya akan efektif jika ia berhasil menipu fungsi heuristik. Dalam kasus ini, meskipun robot berada di dekat jebakan, heuristik tidak tertipu karena tujuannya berada jauh dari jebakan tersebut. Skenario ini memperkuat argumen bahwa dalam banyak kasus non-jebakan, perbedaan utama antara algoritma adalah pada efisiensi, di mana A* menawarkan keseimbangan terbaik antara kecepatan dan keandalan.

IV. Kesimpulan

Berdasarkan serangkaian simulasi dan analisis yang telah dilakukan, makalah ini menarik beberapa kesimpulan penting mengenai penerapan algoritma pencarian rute untuk robot layanan di gedung bertingkat.

1. Uniform Cost Search (UCS) secara konsisten berfungsi sebagai standar untuk optimalitas. Ia selalu berhasil menemukan rute dengan total biaya terendah. Namun, jaminan ini harus dibayar mahal dengan efisiensi komputasi yang sangat rendah, terbukti dari jumlah simpul dieksplorasi yang sangat tinggi di semua skenario. Hal ini membuat UCS tidak praktis untuk aplikasi *real-time* skala besar.
2. Greedy Best-First Search (Greedy BFS) adalah yang tercepat dalam hal waktu komputasi dan efisiensi pencarian. Namun, performanya sangat tidak dapat diandalkan. Meskipun dalam beberapa kasus ia dapat menemukan rute optimal, ia sangat rentan terhadap "jebakan" topologis, di mana estimasi heuristik tidak sejalan dengan biaya rute sebenarnya. Kegagalannya dalam skenario jebakan menunjukkan bahwa algoritma ini terlalu berisiko untuk aplikasi yang menuntut keandalan.
3. Algoritma A* terbukti menjadi pilihan yang paling unggul dan seimbang. Ia berhasil menggabungkan keunggulan dari kedua algoritma lainnya: menemukan rute yang dijamin optimal seperti UCS, namun dengan efisiensi pencarian yang tinggi mendekati Greedy BFS. Kemampuannya untuk tidak terjebak dalam skenario yang menipu sambil tetap menjaga efisiensi komputasi menjadikannya strategi yang paling robust dan efektif untuk masalah navigasi robot yang kompleks.

Secara keseluruhan, penelitian ini menegaskan bahwa untuk membangun sistem robotik yang cerdas dan efisien, pemilihan algoritma yang dapat menyeimbangkan antara kualitas solusi dan kecepatan komputasi, seperti A*, adalah faktor yang krusial.

Tautan Repository Github

Akses program melalui pranala berikut :
https://github.com/iannn23/SimulasiNavigasi_LayananRobotPesananGedungBertingkat.git

Ucapan Terima Kasih

Puji dan syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa. Karena berkat, Rahmat dan karunia-Nya sehingga penulisa dapan menyelesaikan makalah yang berjudul “Simulasi Strategi Navigasi Layanan Robot Pengantar Pesanan di Gedung Bertingkat Menggunakan Algoritma UCS, Greedy Best-First Search dan A*”. Dalam penyusunan makalah ini, penulis tidak luput dari berbagai kesulitan dan hambatan, namun atas bantuan dan dorongan dari berbagai pihak akhirnya makalah ini dapat terselesaikan. Untuk itu, pada kesempatan ini penulis ingin menyampaikan rasa terima kasih yang sebesar-besarnya dan penghargaan yang setinggi-tingginya kepada semua pihak yang telah membantu serta mendukung penulis dalam menyusun dan menyelesaikan makalah ini, yaitu kepada :

1. Bapak Dr. Rinaldi Munir, Bapak Monterico Adrian, M.T, dan Ibu Dr. Nur Ulfa Maulidevi sebagai dosen pengajar mata kuliah IF2211 Strategi Algoritma atas pengajaran materi-materi yang telah dibagikan di kelas Teknik Informatika Semester II Tahun 2024/2025.
2. Para penulis yang telah menciptakan karya-karya yang menjadi landasan bagi penulisan makalah ini. Referensi dari jurnal dan artikel-artikel telah memberikan wawasan dan kontribusi penting pada pemahaman topik.

3. Untuk semua pihak yang tidak dapat disebutkan satu persatu, yang secara langsung maupun tidak langsung telah membantu untuk menyelesaikan makalah ini.

Sekali lagi penulis sampaikan terima kasih atas segala bantuan dan dukungan yang ada. Makalah ini tidak mungkin terwujud tanpa dukungan dan kontribusi dari setiap individu yang disebutkan di atas. Semoga makalah ini dapat menjadi manfaat bagi orang lain yang membacanya.

REFERENCES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

[2] F. R. K. Chung, *Spectral Graph Theory*. Providence, RI: American Mathematical Society, 1997.

[3] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. London, UK: Pearson, 2020.

[4] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100-107, July 1968.

[5] Munir, Rinaldi. "*Graf Bagian 1 – Matematika Diskrit"*, Program Studi Teknik Informatika, STEI ITB, 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf.

[6] N. U. Maulidevi and R. Munir, "*Route Planning (2025) – Bagian 2: Algoritma A∗"*, Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, STEI ITB, 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-(2025)-Bagian2.pdf

[7] N. U. Maulidevi and R. Munir, "*Route Planning (2025) – Bagian 1: BFS, DFS, UCS, Greedy Best First Search".* Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, STEI ITB, 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf

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, 24 Juni 2025

Sebastian Enrico Nathanael
13523134