Penerapan Algoritma A* Untuk Pencarian Rute Terpendek Dalam Navigasi GPS

Muhammad Roihan - 13522152
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13522152@std.stei.itb.ac.id

***Abstract*—Algoritma A* merupakan salah satu algoritma pencarian jalur yang paling efektif dan efisien karena menggabungkan pendekatan heuristik dengan pencarian berbasis graf. Dalam konteks navigasi GPS, kebutuhan untuk menemukan rute terpendek dengan cepat dan akurat sangatlah krusial. Makalah ini akan membahas mengenai penerapan algoritma A* dalam pencarian rute terpendek dan membandingkannya dengan hasil pencarian rute terpendek menggunakan navigasi GPS.**

Keywords—A*, rute terpendek, GPS, Heuristik

I. PENDAHULUAN

Navigasi GPS telah menjadi bagian penting dalam kehidupan sehari-hari, membantu pengguna menemukan rute terpendek dan tercepat ke tujuan mereka. Dengan perkembangan teknologi dan meningkatnya kompleksitas jaringan jalan, kebutuhan akan algoritma pencarian rute yang efisien dan efektif semakin mendesak. Salah satu algoritma yang sering digunakan untuk tujuan ini adalah algoritma A*. Algoritma A* dikenal karena kemampuannya menggabungkan keakuratan dari algoritma Dijkstra dengan efisiensi dari algoritma berbasis heuristik.

Algoritma A* bekerja dengan memanfaatkan fungsi heuristik untuk memperkirakan biaya terendah dari titik awal ke tujuan, sehingga dapat mempercepat proses pencarian rute. Fungsi heuristik ini memberikan estimasi yang informatif tentang jarak atau biaya tersisa ke tujuan, yang memungkinkan algoritma untuk fokus pada jalur yang lebih menjanjikan. Dalam konteks navigasi GPS, penerapan algoritma ini diharapkan dapat meningkatkan kinerja sistem dalam memberikan rute terpendek kepada pengguna, baik dari segi kecepatan maupun akurasi.

Makalah ini bertujuan untuk mengevaluasi penerapan algoritma A* dalam pencarian rute terpendek pada sistem navigasi GPS. Selain itu, makalah ini juga akan membandingkan kinerja algoritma A* dengan hasil pencarian rute menggunakan aplikasi navigasi GPS. Evaluasi dilakukan melalui serangkaian eksperimen yang melibatkan berbagai skenario jaringan jalan dengan tingkat kompleksitas dan ukuran yang berbeda-beda. Eksperimen ini dirancang untuk menguji kecepatan dan akurasi algoritma dalam menemukan rute terpendek di berbagai kondisi.

Hasil dari penelitian ini diharapkan dapat memberikan wawasan yang lebih mendalam mengenai keunggulan dan kelemahan algoritma A*, serta faktor-faktor yang mempengaruhi kinerjanya dalam konteks navigasi GPS. Dengan demikian, temuan ini diharapkan dapat berkontribusi pada pengembangan sistem navigasi yang lebih canggih, efisien, dan responsif terhadap kebutuhan pengguna. Penelitian ini juga bertujuan untuk mengidentifikasi area di mana algoritma A* dapat dioptimalkan lebih lanjut atau digabungkan dengan pendekatan lain untuk meningkatkan kinerja dalam pencarian rute terpendek.

Secara keseluruhan, penelitian ini menargetkan untuk memberikan kontribusi signifikan pada bidang ilmu komputer dan teknik, khususnya dalam aplikasi algoritma pencarian rute terpendek pada sistem navigasi GPS, yang pada akhirnya dapat meningkatkan pengalaman pengguna dan efisiensi transportasi.

II. LANDASAN TEORI

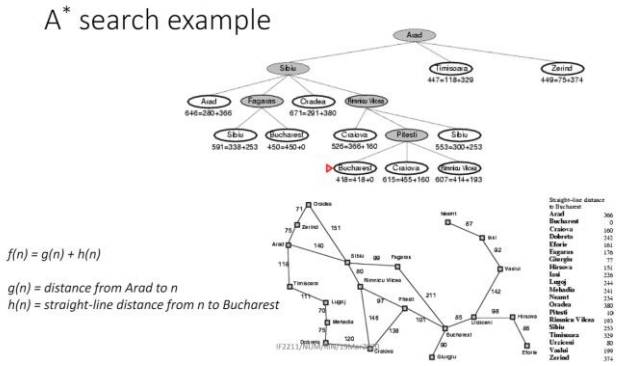
A. *Algoritma A**

Algoritma A* merupakan jenis algoritma pencarian jalur yang dikembangkan untuk menyelesaikan masalah lintas jalur dalam berbagai bidang seperti kecerdasan buatan, robotika, dan permainan komputer. Landasan teorinya didasarkan pada konsep dari dua metode pencarian yang lebih sederhana, yaitu pencarian terurut (seperti algoritma Dijkstra) dan pencarian heuristik. Dalam algoritma A*, setiap simpul dalam ruang pencarian dianggap sebagai node yang terhubung dalam suatu graf. Algoritma ini mencari jalur terpendek dari simpul awal menuju simpul tujuan dengan mengevaluasi setiap simpul yang mungkin dilalui.

Algoritma A* menggabungkan dua nilai dalam mengevaluasi setiap simpul, yaitu biaya aktual dari simpul awal hingga simpul saat ini (g(n)) dan estimasi biaya dari simpul saat ini hingga simpul tujuan (h(n)). Fungsi nilai gabungan f(n) adalah jumlah dari kedua nilai ini, dan digunakan untuk menentukan urutan eksplorasi simpul selanjutnya. Dengan mempertimbangkan kedua faktor ini, A* dapat memilih jalur terpendek dengan meminimalkan jumlah simpul yang dievaluasi.

Kelebihan utama A* adalah kemampuannya untuk menemukan solusi optimal dengan meminimalkan biaya total, asalkan fungsi heuristiknya konsisten. Algoritma ini juga lebih

efisien dibandingkan dengan pencarian blind seperti BFS atau DFS karena menggunakan informasi heuristik untuk memandu pencarian. Namun, kualitas solusi A* sangat bergantung pada kualitas fungsi heuristik yang digunakan, sehingga fungsi heuristik yang buruk dapat menghasilkan solusi yang tidak optimal atau kinerja yang buruk. Kompleksitas waktu A* juga dapat menjadi tinggi terutama pada graf yang besar atau ketika fungsi heuristik memiliki kompleksitas tinggi. Meskipun demikian, A* cocok digunakan ketika solusi optimal diperlukan dan informasi heuristik tersedia atau dapat diperkirakan dengan baik, terutama dalam menemukan jalur terpendek dalam graf yang besar. Namun, jika fungsi heuristik yang baik tidak tersedia atau kompleksitas waktu menjadi kendala, algoritma lain seperti GBFS atau UCS mungkin lebih cocok untuk digunakan. Dengan mempertimbangkan kelebihan, kekurangan, serta kompleksitas waktu dan ruang A*, kita dapat menentukan apakah algoritma ini sesuai untuk masalah pencarian yang dihadapi, dengan memperhatikan kualitas solusi yang diinginkan dan ketersediaan informasi heuristik.



Gambar 2.1 Penerapan Algoritma A*

Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf

B. *Manhattan Distance*

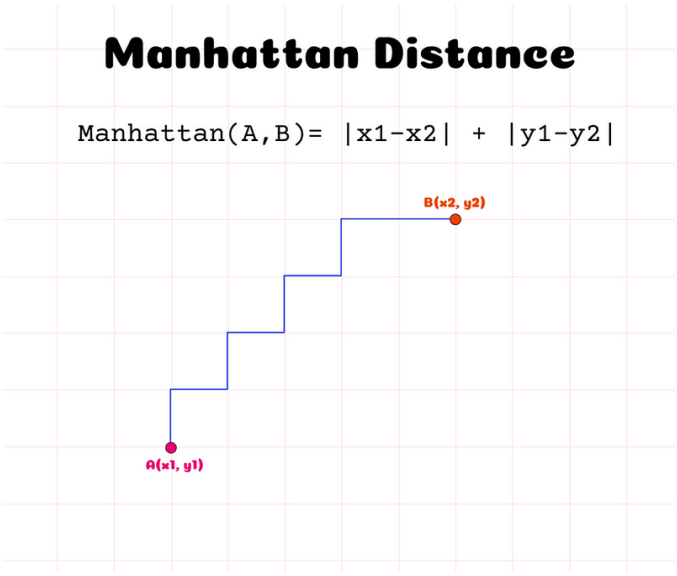
Manhattan distance, juga dikenal sebagai jarak kota blok, adalah metrik jarak yang digunakan dalam berbagai bidang seperti pemodelan pergerakan dalam game, pemecahan masalah lintas jalur, dan kecerdasan buatan. Konsep dasarnya sederhana: untuk mengukur jarak antara dua titik dalam grid, kita hanya menghitung jumlah langkah horizontal dan vertikal yang diperlukan untuk mencapai satu titik dari yang lain, tanpa memperhatikan jarak diagonal.

Dalam pemecahan masalah lintas jalur, Manhattan distance sering digunakan sebagai heuristik dalam algoritma pencarian seperti A*. Heuristik adalah fungsi perkiraan biaya dari titik saat ini hingga titik tujuan. Dalam kasus Manhattan distance, kita menggunakan jumlah langkah horizontal dan vertikal yang diperlukan untuk mencapai titik tujuan dari titik saat ini sebagai perkiraan biaya.

Manhattan distance sangat berguna dalam situasi di mana pergerakan hanya diperbolehkan sepanjang grid, seperti dalam peta jalan kota atau ruang kotak pada papan permainan. Jarak ini dapat dengan mudah dihitung dengan mengambil selisih antara koordinat x dan koordinat y dari dua titik dan menjumlahkannya.

Meskipun sederhana, Manhattan distance memberikan perkiraan yang cukup baik tentang jarak sebenarnya dalam grid, terutama ketika diterapkan dalam algoritma pencarian yang efisien seperti A*. Meskipun ada batasan dalam penggunaannya, terutama ketika ada hambatan atau rintangan dalam grid yang tidak dapat dilewati, Manhattan distance tetap menjadi heuristik yang populer dan berguna dalam berbagai konteks pemodelan pergerakan dan pemecahan masalah lintas jalur.

Dalam penentuan fungsi heuristik dalam algoritma A*, penting untuk memastikan bahwa fungsi heuristik adalah admissible (dapat diterima). Suatu fungsi heuristik dikatakan admissible jika perkiraan biaya yang diberikannya tidak pernah melebihi biaya sebenarnya untuk mencapai tujuan. Dengan kata lain, heuristik harus memberikan estimasi yang konservatif tentang biaya terpendek yang diperlukan untuk mencapai tujuan. Sebagai contoh, Manhattan distance sering digunakan sebagai heuristik dalam algoritma A*, karena jarak yang dihitung dengan menggunakan Manhattan distance tidak pernah melebihi jarak sebenarnya dalam grid, sehingga memenuhi kriteria admissibility. Ini membuatnya menjadi pilihan heuristik yang baik karena memberikan estimasi yang akurat namun efisien dalam menavigasi ruang pencarian.



Gambar 2.2 Contoh Manhattan Distance

Sumber:https://aigents.co/data-science-blog/publication/distance-metrics-for-machine-learning

III. Implementasi dan Pembahasan

Dalam menyelesaikan permasalahan ini penulis membuat program berbasis OOP dalam bahasa python yang mengimplementasikan kinerja algoritma A* yang menggunakan manhattan distance sebagai heuristiknya.

A. Kelas Graph

Kelas Graph digunakan untuk merepresentasikan graf berarah dengan bobot, di mana setiap simpul (vertex) memiliki nilai heuristik yang dihitung menggunakan Manhattan Distance. Kelas ini menyediakan metode untuk menambah simpul dan sisi (edge) di dalam graf.

```
class Graph:
    def __init__(self):
        self.vertices = {}
        self.manhattan = {}
```

Sumber: Dokumen Penulis

Metode inisialisasi digunakan untuk membuat instance baru dari kelas Graph. Ketika objek Graph baru dibuat, dua atribut utama diinisialisasi:

- vertices: Sebuah dictionary yang menyimpan simpul-simpul graf dan daftar tetangga mereka beserta bobot tepi yang menghubungkan mereka.
- manhattan: Sebuah dictionary yang menyimpan nilai heuristik Manhattan Distance untuk setiap simpul.

```
def add_vertex(self, vertex, manhattan):
    if vertex not in self.vertices:
        self.vertices[vertex] = []
    self.manhattan[vertex] = manhattan
```

Sumber : Dokumen Penulis

Metode add_vertex digunakan untuk menambahkan simpul baru ke dalam graf.

- Parameter vertex adalah nama atau label dari simpul yang akan ditambahkan.
- Parameter manhattan adalah nilai heuristik Manhattan Distance untuk simpul tersebut.

Jika simpul belum ada di dalam dictionary vertices, simpul tersebut ditambahkan sebagai kunci dengan nilai berupa list kosong (untuk menyimpan tetangga dan bobot tepi). Nilai heuristik juga disimpan dalam dictionary manhattan.

```
def add_edge(self, vertex1, vertex2, cost):
    if vertex1 in self.vertices and vertex2 in self.vertices:
    self.vertices[vertex1].append((vertex2, cost))
```

Sumber : Dokumen Penulis

Metode add_edge digunakan untuk menambahkan sisi berarah dari satu simpul ke simpul lainnya.

- Parameter vertex1 adalah simpul asal dari sisi yang akan ditambahkan.
- Parameter vertex2 adalah simpul tujuan dari sisi yang akan ditambahkan.
- Parameter cost adalah bobot atau biaya yang terkait dengan sisi tersebut.

*B. Algoritma A**

```
def a_star(self, start, goal):
    open_set = []
    heapq.heappush( open_set, (0, start))
    came_from = {}
    g_score = {
        vertex: float("inf") for vertex in self.vertices
    } # Biaya sejauh ini dari titik start ke setiap node
    g_score[start] = 0
    f_score = {
        vertex: float("inf") for vertex in self.vertices
    } # Estimasi biaya dari titik start ke node goal + manhattan distances
    f_score[start] = self.manhattan[start]
```

Sumber : Dokumen Penulis

- open_set: Antrian prioritas (heap) yang menyimpan simpul-simpul yang akan dieksplorasi, diurutkan berdasarkan nilai f_score.
- came_from: Dictionary yang melacak simpul sebelumnya untuk setiap simpul, yang membantu merekonstruksi jalur setelah tujuan tercapai.
- g_score: Dictionary yang menyimpan biaya terendah dari simpul start ke setiap simpul.
- f_score: Dictionary yang menyimpan perkiraan biaya terendah dari simpul start ke goal melalui simpul tertentu (menggunakan heuristik Manhattan Distance).

```
while open_set:
    current_f, current = heapq.heappop(
        open_set
    ) # Ambil node dengan nilai f terendah
    if current == goal:
        path = []
        while current in came_from:
            path.append(current)
            current = came_from[current]
        path.append(start)
        return path[::-1]
    # Balikkan jalur karena telah dimulai dari goal
```

Sumber : Dokumen Penulis

- Selama open_set tidak kosong, simpul dengan f_score terendah diambil dari antrian.
- Jika simpul saat ini (current) adalah tujuan, jalur dikonstruksi dengan melacak kembali dari goal ke start

menggunakan came_from, lalu mengembalikan jalur yang ditemukan..

```
    for neighbor, cost in self.vertices[current]:
        if g_score[current] + cost + self.manhattan[neighbor] < f_score[neighbor]:
            came_from[neighbor] = current
            g_score[neighbor] = g_score[current] + cost
            f_score[neighbor] = (
                g_score[current] + cost + self.manhattan[neighbor]
            )
            if neighbor not in [node[1] for node in open_set]:
                heapq.heappush(open_set, (f_score[neighbor], neighbor))
                # Jika neighbor tidak ada di open_set, masukkan ke open_set
            else:
                self.remove_from_open_set(open_set, neighbor)
                heapq.heappush(open_set, (f_score[neighbor], neighbor))
                # Jika neighbor ada di open_set tetapi f_scorenya lebih kecil dibanding
                # yang ada di openset, ganti neighbor dengan yang baru

    return None  # Jika tidak ada jalur yang ditemukan
```

Gambar 3.1 Ekspansi Tetangga

Sumber : Dokumen Penulis

- Untuk setiap tetangga (neighbor) dari simpul saat ini, biaya baru (tentative_g_score) dihitung.
- Jika biaya baru lebih rendah dari f_score tetangga saat ini, simpul tersebut diperbarui:
    - came_from: Tetangga sekarang berasal dari simpul saat ini.
    - g_score: Biaya dari simpul awal ke tetangga diperbarui.
    - f_score: g_score ditambah nilai heuristik (Manhattan Distance) diperbarui.Selama open_set tidak kosong, simpul dengan f_score terendah diambil dari antrian.
- Tetangga ditambahkan ke open_set jika belum ada, atau open_set diperbarui jika tetangga sudah ada di dalamnya.

C. *Analisis Dan Pengujian*

Untuk pengujian pertama penulis melakukan pencarian rute dengan input seperti berikut.

```
graph.add_vertex("Boemi Mitoha", 375.81)
graph.add_vertex("Bellamie Boulangerie", 226.48)
graph.add_vertex("Shabu Hachi", 445.07)
graph.add_vertex("Bonfire", 0)
graph.add_edge("Boemi Mitoha", "Bellamie Boulangerie", 550)
graph.add_edge("Boemi Mitoha", "Shabu Hachi", 180)
graph.add_edge("Shabu Hachi", "Bellamie Boulangerie", 400)
graph.add_edge("Shabu Hachi", "Bonfire", 700)
graph.add_edge("Bellamie Boulangerie", "Bonfire", 270)
start = "Boemi Mitoha"
goal = "Bonfire"
```

Gambar 3.2 Input Test Case 1

Sumber : Dokumen Penulis

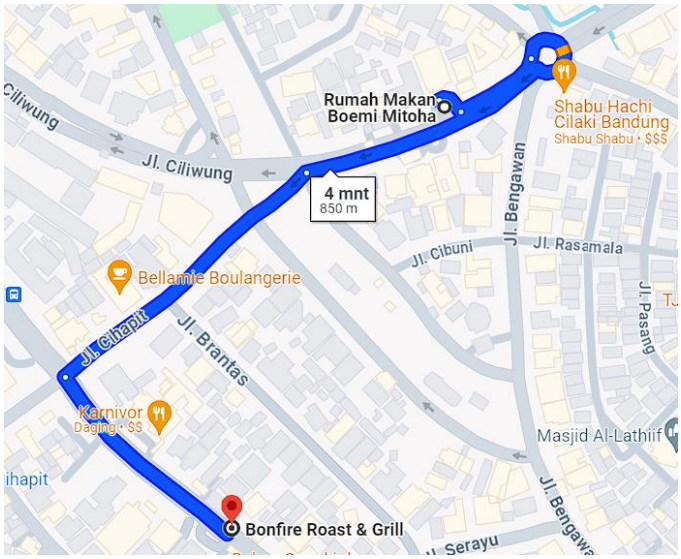
Didapatkan hasil seperti berikut.

```
Shortest Path from Boemi Mitoha to Bonfire:
Boemi Mitoha -> Bellamie Boulangerie -> Bonfire
```

Gambar 3.3 Output Test Case 1

Sumber : Dokumen Penulis

Hasil tersebut sesuai dengan hasil yang didapatkan menggunakan navigasi GPS.



Gambar 3.4 Pengujian Test Case 1 Menggunakan Navigasi GPS

Sumber : Dokumen Penulis

Dapat dilihat pada gambar 3.4 navigasi GPS lebih mengarahkan jalur melalui Jalan Cipahit karena lebih dekat dibandingkan melalui Jalan Bengawan.

Dari percobaan tersebut, kita dapat menyimpulkan bahwa algoritma A* merupakan algoritma yang efektif untuk pencarian jalur terpendek. Namun, terdapat kelemahan pada algoritma ini karena rute yang diberikan oleh GPS sering kali mempertimbangkan faktor lain selain jarak. Oleh karena itu, kita tidak dapat selalu mencari rute paling optimal hanya dengan mencari rute terpendek berdasarkan jarak.

Menyadari hal ini, penulis mengeksplorasi alternatif heuristik untuk algoritma A* guna mempertimbangkan faktor-faktor lain selain jarak, untuk menemukan rute yang benar-benar optimal dalam berbagai situasi.

1. Mempertimbangkan jarak dan kondisi lalu lintas

Faktor lain yang penting dipertimbangkan adalah kondisi lalu lintas. Sebagai contoh, perhatikan ilustrasi berikut.

- A ke B: 5000 m, mengalami kemacetan dengan waktu tempuh 30 menit
- A ke C: 8000 m, lancar dengan waktu tempuh 10 menit
- B ke D: 10000 m, lancar dengan waktu tempuh 20 menit
- C ke D: 10000 m, lancar dengan waktu tempuh 20 menit

Meskipun jarak dari A ke B lebih dekat dibandingkan A ke C, waktu tempuh dari A ke B jauh lebih lama karena

kemacetan. Dengan mempertimbangkan kondisi lalu lintas, rute A – C – D dipilih karena total waktu tempuhnya adalah 30 menit (10 menit dari A ke C dan 20 menit dari C ke D), dibandingkan dengan 50 menit melalui rute A – B – D (30 menit dari A ke B dan 20 menit dari B ke D).

Dari ilustrasi tersebut, kita dapat melihat bahwa mempertimbangkan kondisi lalu lintas dalam heuristik dapat menghasilkan rute yang lebih optimal dalam hal waktu tempuh. Namun, penting untuk memastikan bahwa heuristik ini tetap admissible agar algoritma A* dapat bekerja secara optimal.dipilih karena bisa jadi rute tersebut memiliki perkiraan waktu tempuh minimum. Namun solusi ini menimbulkan pertanyaan baru apakah heuristik yang digunakan *admissible* atau tidak.

Misalkan jalan dari C ke D lurus. Hal ini berarti jika kita menggunakan heuristik manhattan distance akan menghasilkan nilai yang sama dengan jarak aktual yaitu 10000 dalam satuan meter. Akan tetapi jika kita tambahkan faktor lain yaitu kondisi lalu lintas yang dalam kasus ini diwakili dengan waktu tempuh maka nilai heuristiknya menjadi 10000 + 20 = 10020. Karena h(n) > h*(n) maka dapat disimpulkan solusi ini tidak admissible.

2. Hanya mempertimbangkan waktu tempuh

Alternatif kedua yang dieksplorasi penulis adalah perhitungan nilai heuristik yang hanya mempertimbangkan waktu tempuh tanpa memperhatikan jarak. Dengan menggunakan waktu tempuh sebagai satu-satunya faktor dalam heuristik, kita dapat fokus pada optimasi rute berdasarkan kondisi lalu lintas dan kecepatan perjalanan, yang seringkali lebih relevan untuk pengguna jalan. Misalnya, sebuah rute mungkin lebih panjang dalam hal jarak tetapi memiliki waktu tempuh yang lebih singkat karena lalu lintas yang lebih lancar atau jalan tol yang lebih cepat.

Alternatif ini kembali mengevaluasi kevalidan heuristik yang digunakan, apakah admissible atau tidak. Dalam contoh sebelumnya, alternatif ini menyajikan sebuah heuristik yang admissible karena h(n) < h*(n) (20 < 10.000). Namun, penulis kemudian menelusuri kondisi di mana heuristik tersebut tidak admissible. Misalnya, jika jarak dari C ke D adalah 20 meter dengan waktu tempuh 25 menit, kondisi ini jelas tidak admissible karena 25 > 20. Namun, jika kita mempertimbangkan kasus tersebut, berarti kecepatan kendaraannya adalah 0.8 meter per menit. Mengingat bahwa kendaraan dengan kecepatan sebegitu lambat tidak mungkin ada, maka dapat disimpulkan bahwa alternatif ini tetap admissible.

Namun, pendekatan ini juga memiliki kelemahan. Dengan mengabaikan jarak sepenuhnya, kita mungkin mengabaikan faktor-faktor lain seperti konsumsi bahan bakar atau keausan kendaraan yang lebih tinggi pada rute yang lebih panjang. Selain itu, waktu tempuh dapat sangat bervariasi tergantung pada kondisi lalu lintas yang dinamis dan sulit diprediksi dengan akurasi tinggi. Oleh karena itu, heuristik yang hanya mempertimbangkan waktu tempuh mungkin tidak selalu memberikan solusi yang paling optimal dalam semua situasi.

IV. KESIMPULAN

Navigasi GPS telah menjadi bagian penting dalam kehidupan sehari-hari, membantu pengguna menemukan rute terpendek dan tercepat ke tujuan mereka. Dengan perkembangan teknologi dan meningkatnya kompleksitas jaringan jalan, kebutuhan akan algoritma pencarian rute yang efisien dan efektif semakin mendesak. Salah satu algoritma yang menonjol dalam menyelesaikan masalah ini adalah algoritma A*.

Berdasarkan penelitian yang dilakukan, A* yang menggunakan heuristik Manhattan Distance memang terbukti sebagai algoritma yang efektif dalam pencarian rute terpendek. Namun, dalam menentukan rute paling efisien dalam navigasi GPS, perlu diperhatikan faktor-faktor lain selain jarak. Oleh karena itu, pendekatan yang diusulkan dalam makalah ini adalah menggunakan heuristik yang mempertimbangkan waktu tempuh tanpa memperhitungkan jarak tempuh.

Meskipun pendekatan ini memiliki potensi untuk memberikan solusi yang optimal dalam hal waktu tempuh, pendekatan ini juga memiliki kelemahan. Dengan mengabaikan jarak sepenuhnya, faktor-faktor lain seperti konsumsi bahan bakar atau keausan kendaraan pada rute yang lebih panjang dapat diabaikan. Selain itu, waktu tempuh sangat dipengaruhi oleh kondisi lalu lintas yang dinamis dan sulit diprediksi dengan akurasi tinggi. Oleh karena itu, heuristik yang hanya mempertimbangkan waktu tempuh mungkin tidak selalu memberikan solusi yang paling optimal dalam semua situasi.

Diperlukan analisis lebih lanjut untuk menyeimbangkan antara jarak dan waktu tempuh dalam heuristik yang digunakan. Hal ini penting agar algoritma A* dapat menemukan rute yang paling efisien secara keseluruhan, dengan mempertimbangkan berbagai faktor yang memengaruhi pengalaman pengguna dan kebutuhan praktis dalam navigasi GPS.

LINK REPOSITORI

https://github.com/mroihn/A_Star_Shortest_Path

V. UCAPAN TERIMA KASIH

Dengan penuh rasa syukur, penulis ingin mengucapkan puji dan syukur kepada Tuhan Yang Maha Esa atas segala rahmat, karunia, serta taufik dan hidayah-Nya, yang telah memandu dan memberikan keberkahan sehingga penulis berhasil menyelesaikan makalah berjudul "Penerapan Algoritma A* Untuk Pencarian Rute Terpendek Dalam Navigasi GPS". Makalah ini disusun sebagai bagian dari tugas mata kuliah Strategi Algoritma IF2211.

Penulis juga mengucapkan terima kasih yang setinggi-tingginya kepada Bapak Ir. Rila Mandala, M.Eng., Ph.D. dan Bapak Monterico Adrian, S.T., M.T., sebagai dosen pengajar dalam mata kuliah Strategi Algorimta IF2211 Kelas K3. Terima kasih atas dedikasi, bimbingan, dan pengajaran yang telah diberikan selama satu semester ini, memberikan kontribusi besar dalam pembentukan pemahaman kami sebagai mahasiswa. Penulis juga beterimakasih kepada seluruh pihak yang telah berkontribusi baik secara langsung maupun tidak

langsung terhadap kelancaran penulisan makalah ini yang namanya tidak bisa disebutkan satu persatu. Penulis juga ingin menyampaikan permohonan maaf apabila terdapat kesalahan dalam penulisan makalah ini. Semoga makalah ini dapat bermanfaat dan memberikan sumbangan positif dalam pemahaman mata kuliah Strategi Algoritma IF2211.

REFERENCES

[1] Rinaldi Munir, "Route Planning Bagia 2". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf diakses pada tanggal 12 Juni 2024.

[2] https://www.geeksforgeeks.org/a-search-algorithm/ diakses pada tanggal 12 Juni 2024.

[3] https://algodaily.com/lessons/what-is-the-manhattan-distance diakses pada tanggal 12 Juni 2024

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, 12 Juni 2024

Muhammad Roihan dan 13522152