Formulasi Algoritma A* Berbasis Jarak Manhattan untuk Penentuan Rute Terpendek Peninjauan Laporan Kerusakan Jalan pada Sebuah Aplikasi Laporan Publik Berbasis Peta

Arghawisesa Dwinanda Arham - 13524100
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: arghawisesa@gmail.com, 13524100@std.stei.itb.ac.id

***Abstract*—Penentuan rute kunjungan yang optimal merupakan masalah krusial dalam operasional lapangan, seperti peninjauan kerusakan infrastruktur jalan. Makalah ini mengusulkan formulasi dan implementasi algoritma A-Star dengan heuristik Jarak Manhattan untuk menemukan rute terpendek pada aplikasi pelaporan publik berbasis peta, yaitu CommunityMap. Algoritma ini mengevaluasi simpul berdasarkan fungsi biaya yang merupakan penjumlahan dari biaya jarak aktual menggunakan formula Haversine dan estimasi sisa jarak menggunakan metrik Manhattan. Pendekatan ini bertujuan mengatasi ketidakefisienan metode pencarian buta maupun kelemahan jebakan optimum lokal pada pencarian serakah. Eksperimen dilakukan pada lingkungan aplikasi web secara waktu-nyata dengan mengambil data koordinat lintang dan bujur dari laporan jalan rusak. Hasil analisis menunjukkan bahwa algoritma secara presisi dan efisien mampu menemukan urutan kunjungan dengan total jarak terpendek. Fungsi heuristik yang diterapkan memenuhi syarat perhitungan batas bawah (admissible), sehingga pencarian rute dijamin optimal.**

Keywords*— *Algoritma A-Star, Pencarian Heuristik, Jarak Manhattan, Jarak Haversine, Optimasi Rute, Pelaporan Publik.

I. PENDAHULUAN

Kerusakan infrastruktur jalan raya berdampak langsung pada kelancaran transportasi dan keselamatan masyarakat. Melalui sistem pelaporan publik, masyarakat dapat melaporkan lokasi jalan berlubang, genangan air, atau lampu jalan yang padam secara presisi menggunakan koordinat GPS. Sistem yang digunakan dalam makalah ini adalah sebuah aplikasi berbasis web bernama CommunityMap, sebuah proyek yang dirancang sebagai platform *crowdsourcing* untuk melaporkan jalan yang bermasalah, di mana nantinya petugas daerah setempat dapat melihat kerusakan jalan tersebut dan dapat memperbaikinya. CommunityMap ini awalnya merupakan proyek yang meliputi Saya (Arghawisesa Dwinanda Arham) mahasiswa Teknik Informatika ITB angkatan 2024, Nashiruddin Akram mahasiswa Teknik Informatika ITB angkatan 2024, dan Rafi Putra Nugraha mahasiswa Sistem dan Teknologi Informasi ITB angkatan 2024, untuk sebuah Tugas Besar mata kuliah ET3204 Layanan Tersambung dan Komputasi Awan. Pengembangan sistem ini kemudian dilanjutkan dan dioptimalisasi untuk memenuhi kebutuhan tugas mata kuliah IF2211 Strategi Algoritma khususnya pada penyelesaian optimasi rute. Namun, banyaknya jumlah laporan yang masuk menjadi tantangan tersendiri bagi pihak berwenang (misalnya Dinas Pekerjaan Umum/DPU) untuk meninjau dan menyelesaikan perbaikan secara efisien.

Apabila petugas lapangan melakukan peninjauan secara acak atau sekadar berdasarkan laporan yang paling lama masuk, mereka berisiko menempuh rute yang bolak-balik melintasi area yang sama. Hal ini mengakibatkan pemborosan waktu tempuh dan konsumsi bahan bakar. Oleh karena itu, diperlukan suatu metode komputasional untuk menentukan urutan titik laporan (node) yang menghasilkan jarak keseluruhan paling minimum.

Masalah pencarian rute terpendek (path/route planning) dapat diselesaikan dengan berbagai algoritma pencarian di dalam domain Kecerdasan Buatan (Artificial Intelligence). Metode pencarian buta (uninformed search) seperti Breadth-First Search (BFS) maupun Uniform Cost Search (UCS) mampu memberikan solusi yang optimal, namun sangat tidak efisien dari sisi penggunaan memori dan waktu komputasi karena jumlah ekspansi simpul yang eksponensial [1]. Sebagai alternatif, metode pencarian heuristik (informed search) mengevaluasi kualitas suatu simpul berdasarkan nilai estimasinya terhadap tujuan.

Algoritma A* (dibaca: A-Star) adalah salah satu algoritma pencarian heuristik paling populer yang memadukan keunggulan UCS dan Greedy Best-First Search. A* memastikan bahwa solusi yang ditemukan dijamin optimal (rute paling pendek) asalkan fungsi heuristik yang dirancang bersifat admissible atau tidak pernah overestimate terhadap biaya sebenarnya [2].

Makalah ini membahas bagaimana formulasi algoritma A* dipadukan dengan heuristik Jarak Manhattan (Manhattan Distance) dan diimplementasikan secara langsung untuk merencanakan rute kunjungan peninjauan jalan secara otomatis melalui peta digital.

II. Dasar Teori

A. Penentuan Rute (Path Planning) dan Pencarian Heuristik

Penentuan rute merupakan persoalan fundamental pencarian lintasan graf yang bertujuan menemukan urutan perjalanan terbaik dari titik awal ke satu atau beberapa titik tujuan [1]. Graf didefinisikan dengan himpunan simpul V yang mewakili lokasi dan busur E yang mewakili jalan dengan bobot/jarak tertentu.

Pencarian heuristik bekerja dengan memanfaatkan informasi ekstra dari luar definisi persoalan (misalnya posisi spasial geometri) untuk mengestimasi kedekatan suatu simpul dengan status tujuan (goal state) [2].

B. Algoritma A*

Algoritma A* merupakan algoritma pencarian jalur yang menghitung estimasi biaya total terendah melalui suatu node n. Algoritma ini menggunakan fungsi evaluasi:

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

Di mana g(n) merepresentasikan biaya aktual (jarak) dari titik awal pencarian hingga ke titik n, dan h(n) merepresentasikan fungsi heuristik, yaitu biaya estimasi terendah dari titik n menuju ke tujuan akhir [2]. A* pasti menemukan solusi jika ada, serta optimal asalkan h(n) dirancang agar bersifat admissible (lebih kecil dari biaya aslinya [2].

C. Formula Jarak Spasial

Haversine Distance digunakan untuk menghitung jarak akurat dua titik di atas permukaan melengkung bumi (bola) menggunakan koordinat lintang (latitude) dan bujur (longitude). Ini merepresentasikan jarak sebenarnya sebagai bagian dari nilai g(n).

Manhattan Distance adalah pengukuran jarak grid ortogonal yang menjumlahkan selisih absolut jarak pada sumbu X dan Y. Formula ini sering digunakan sebagai heuristik h(n) pada persoalan spasial dua dimensi, seperti persoalan 8-puzzle atau pergerakan yang membentuk grid perkotaan [2].

III. Metodologi

A. Arsitektur dan Alur Kerja Sistem

Sistem perencana rute diimplementasikan di atas aplikasi web React (Next.js) dan layanan basis data pelaporan. Langkah-langkah penyelesaian A* adalah sebagai berikut:

1) *Representasi Input*: Posisi awal petugas ditentukan secara fleksibel via klik pada map interaktif. Beberapa laporan warga dengan status aktif (Baru, Terverifikasi, Dalam Proses) dipilih sebagai titik waypoints yang wajib dikunjungi.

2) *Kondisi Goal*: Pencarian berhenti jika seluruh node yang dipilih oleh admin telah tercatat di dalam urutan lintasan yang valid.

3) *Ekspansi Simpul (Open List)*: Algoritma menggunakan struktur data antrean prioritas (Priority Queue) untuk menyimpan rute-rute kandidat (simpul hidup).

4) *Representasi State Space dan Closed List*: Karena persoalan rute peninjauan ini pada dasarnya adalah variasi dari Traveling Salesperson Problem (TSP), representasi status (state) dari setiap simpul di dalam pohon pencarian tidak hanya berdasarkan posisi (koordinat) saat ini, melainkan kombinasi dari lokasi saat ini dan himpunan laporan yang belum dikunjungi. Di dalam implementasi, state key diformulasikan sebagai string gabungan currentId | unvisitedIds. Hal ini bertujuan untuk menghindari duplikasi komputasi (penghindaran siklus). Setiap kali sebuah state dibangkitkan, program akan mengecek struktur data Map (sebagai closed list). Jika state dengan sisa laporan yang sama persis sudah pernah diekspansi sebelumnya dengan biaya g(n) yang lebih kecil atau sama, maka state tersebut akan diabaikan (pruning), sehingga algoritma terhindar dari infinite loop dan pencarian berulang yang tidak perlu.

B. Alasan Pemilihan Algoritma

Dalam menyelesaikan persoalan urutan peninjauan laporan, pendekatan intuitif yang paling sederhana sebenarnya adalah menggunakan algoritma heuristik Greedy (strategi Nearest Neighbor). Pada pendekatan Greedy, sistem akan menginstruksikan petugas untuk selalu bergerak ke titik laporan terdekat dari lokasinya saat ini tanpa memedulikan rute secara keseluruhan. Meskipun algoritma Greedy berjalan sangat cepat dengan kompleksitas waktu polinomial, pendekatan ini memiliki kelemahan fatal yaitu rentan terjebak pada optimum lokal (local optimum trap). Sebagai contoh skenario, jika terdapat kumpulan laporan di wilayah Jakarta dan satu laporan terisolasi di Bandung, algoritma Greedy mungkin akan membuat petugas berputar-putar menghabiskan rute di Jakarta terlebih dahulu, dan menyisakan Bandung di akhir, yang pada akhirnya dapat menghasilkan total jarak tempuh (tour) yang lebih jauh jika petugas harus kembali ke titik awal. Oleh karena itu, sistem ini mengimplementasikan algoritma A*. Berbeda dengan Greedy yang hanya melihat fungsi heuristik jarak terdekat h(n) saja, algoritma A* mengevaluasi fungsi f(n) = g(n) + h(n). Keberadaan nilai g(n) (akumulasi jarak yang sudah ditempuh sejak dari titik awal) berfungsi sebagai penyeimbang. Jika suatu rute mulai terlihat menyimpang terlalu jauh dari arah penyelesaian global, nilai g(n) akan membengkak, sehingga algoritma A* akan secara cerdas menunda ekspansi rute tersebut dan mengeksplorasi alternatif rute lain di dalam antrean prioritas. Hal ini menjamin bahwa solusi yang dihasilkan oleh aplikasi bukanlah sekadar rute asal sampai, melainkan solusi absolute optimal secara matematis.

C. Formulasi Jarak Aktual g(n)

Dalam implementasi ini, pergerakan tidak diukur berdasarkan piksel semu melainkan koordinat GPS faktual. Perhitungan g(n) di setiap simpul mengkalkulasi akumulasi jarak Haversine dari titik petugas awal (Start) melewati seluruh simpul laporan sebelumnya, hingga simpul saat ini. Perhitungan jarak tidak dapat dilakukan menggunakan rumus jarak Euclidean planar biasa (teorema Pythagoras). Penggunaan Euclidean pada skala peta dunia nyata akan menghasilkan galat (error) yang signifikan karena mengabaikan kelengkungan bentuk bumi.

Untuk mendapatkan nilai g(n) yang presisi antar dua titik koordinat, sistem dalam aplikasi ini menggunakan perhitungan trigonometri bola (spherical trigonometry) yang dikenal sebagai Formula Haversine [3]. Formula ini menghitung jarak lingkaran besar (great-circle distance) antara dua titik pada permukaan bola berdasarkan garis bujur dan lintangnya. Persamaan matematis yang diimplementasikan dalam sistem adalah sebagai berikut:

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$g(n) = d = R \cdot c$$

φ adalah lintang (latitude) dalam radian. λ adalah bujur (longitude) dalam radian. *R* adalah radius rata-rata bumi (sekitar 6.371 km).

Penggunaan formula Haversine ini memastikan bahwa setiap bobot sisi (garis antar simpul) di dalam graf ruang pencarian (state space) merepresentasikan jarak udara nyata secara akurat. Akurasi nilai g(n) ini sangat krusial, karena jika nilai g(n) cacat, maka penjumlahan f(n) = g(n) + h(n) pada algoritma A^* juga akan menjadi tidak valid dan dapat mengarah pada keputusan rute yang salah kaprah.

D. *Formulasi Fungsi Heuristik h(n)*

Fungsi heuristik berperan mempersempit ruang pencarian ke arah simpul-simpul yang berpotensi memiliki rute sisa terpendek. Karena kita ingin meninjau seluruh laporan, heuristik h(n) dihitung dengan menjumlahkan Manhattan Distance sisa titik laporan yang belum terkunjungi. Hal ini merepresentasikan prediksi logis jarak total geografis yang harus dihabiskan untuk menutup rute.

E. *Struktur Data Antrean Prioritas (Min-Heap)*

Salah satu komponen paling krusial dalam algoritma A* adalah proses pengambilan simpul dengan nilai f(n) terkecil pada open list. Jika menggunakan array biasa dan melakukan sorting pada setiap iterasi, algoritma akan berjalan sangat lambat dengan kompleksitas O(VlogV) di setiap langkahnya. Oleh karena itu, implementasi pada sistem ini membangun struktur data Priority Queue secara mandiri menggunakan prinsip Min-Heap (Binary Tree yang direpresentasikan dalam array). Saat rute baru ditambahkan (push), sistem melakukan operasi bubble-up untuk menaikkan elemen jika prioritasnya lebih kecil dari induknya, yang berjalan dalam kompleksitas waktu O(logK) di mana K adalah ukuran heap. Sebaliknya, saat mengambil simpul hidup dengan f(n) terkecil (pop), sistem memindahkan elemen terakhir ke akar dan melakukan operasi sink-down (membandingkan dengan anak-anaknya dan menukarnya dengan yang terkecil) yang juga berjalan dalam waktu O(logK). Penggunaan Min-Heap ini secara drastis mengoptimalkan waktu eksekusi komputasi pencarian rute pada aplikasi.

IV. Hasil dan Pembahasan

A. *Implementasi Pencarian Ruang Status dan Pemrosesan Koordinat*

Algoritma diimplementasikan di lingkungan backend menggunakan Node.js. Rute ini diformulasikan sebagai masalah Traveling Salesperson Problem (TSP) dengan menggunakan objek peta spasial dua dimensi. Komputasi menggunakan dua formula jarak, yaitu Haversine Distance untuk mengukur kelengkungan bumi (skala nyata) dan Manhattan Distance sebagai pendekatan perkiraan blok kota (heuristik).

Secara teknis, setiap koordinat laporan memiliki dua data masuk utama, yaitu titik lintang (latitude atau lat) dan titik bujur (longitude atau lng).

1) *Perhitungan Haversine Distance*

Pencarian nilai g(n) dihitung dengan mencari jarak melengkung aktual di bumi dari koordinat awal menuju koordinat berikutnya. Nilai lintang dan bujur diubah ke dalam radian sebelum dikomputasi ke dalam rumus Haversine sebagai berikut:

```
function haversineDistance(lat1, lng1, lat2, lng2) {
  const EARTH_RADIUS_KM = 6371; // Radius Bumi
  const toRad = (deg) => (deg * Math.PI) / 180;
  const dLat = toRad(lat2 - lat1);
  const dLng = toRad(lng2 - lng1);
  const a = Math.sin(dLat / 2) ** 2 +
            Math.cos(toRad(lat1)) *
Math.cos(toRad(lat2)) * Math.sin(dLng / 2) ** 2;

  // Mengembalikan jarak metrik dalam Kilometer
  return EARTH_RADIUS_KM * 2 * Math.atan2(Math.sqrt(a),
Math.sqrt(1 - a));
}
```

2) *Perhitungan Manhattan Distance*

Untuk mencari nilai admissible heuristik (h(n)), digunakan selisih vertikal dan horizontal sederhana berdasarkan bujur dan lintang, namun disesuaikan (dikonversi) dengan besaran kilometer:

```
function manhattanDistance(lat1, lng1, lat2, lng2) {
  const KM_PER_DEGREE_LAT = 111.32;
  const avgLat = (lat1 + lat2) / 2;

  // Konversi longitude menyempit berdasarkan
lintangnya (garis bumi)
  const kmPerDegreeLng = KM_PER_DEGREE_LAT *
Math.cos((avgLat * Math.PI) / 180);

  const dLat = Math.abs(lat2 - lat1) *
KM_PER_DEGREE_LAT;
  const dLng = Math.abs(lng2 - lng1) * kmPerDegreeLng;

  return dLat + dLng;
}
```

3) *Perhitungan Algoritma A**

Setiap status (state) pada A* didefinisikan sebagai kombinasi antara titik koordinat saat ini dan himpunan id laporan yang belum dikunjungi. Karena komputasi bertujuan mengunjungi seluruh laporan yang dicentang admin, nilai Heuristik dievaluasi menggunakan Max Manhattan Distance ke setiap titik yang masih tersisa. Berikut adalah pemrosesan graf pencariannya:

```
// Ekspansi ke semua node yang belum dikunjungi
(Looping Unvisited Set)
```

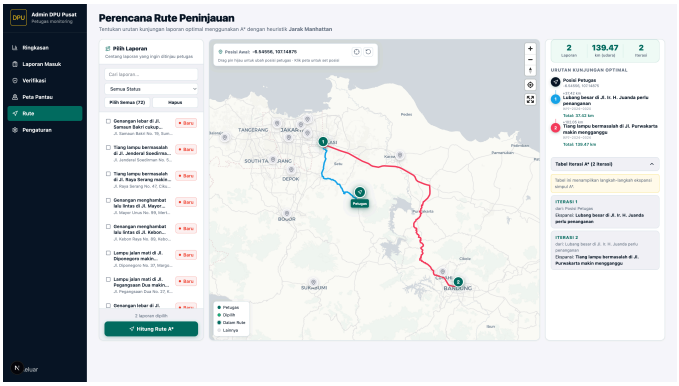
```
for (const nextId of current.unvisitedIds) {
  const nextReport = reports.find((r) => r.id ===
nextId);
  const newUnvisitedIds =
current.unvisitedIds.filter((id) => id !== nextId);

  // Menghitung g(n) komulatif baru = g(n) lama + Cost
Haversine (Langkah ini)
  const stepCost = haversineDistance(
    current.currentPos.lat, current.currentPos.lng,
    nextReport.lat, nextReport.lng
  );
  const newG = current.g + stepCost;

  // Menghitung h(n) baru = Jarak manhattan terjauh
dari nextReport ke node unvisited lainnya
  let newH = 0;
  if (newUnvisitedIds.length > 0) {
    newH = Math.max(
      ...newUnvisitedIds.map((id) => {
        const r = reports.find((x) => x.id === id);
        return manhattanDistance(nextReport.lat,
nextReport.lng, r.lat, r.lng);
      })
    );
  }

  const newF = newG + newH; // Kalkulasi fungsi f(n)

  // Desain state untuk Queue dan Deduplikasi
  const nextState = {
    currentId: nextReport.id,
    currentPos: { lat: nextReport.lat, lng:
nextReport.lng },
    unvisitedIds: newUnvisitedIds,
    g: newG,
    h: newH,
    f: newF,
    // ... data logging lainnya disembunyikan untuk
simplifikasi
  };

  // Memasukkan node yang telah diekspansi dan dihitung
kembali ke dalam Antrean Prioritas
  pq.push(nextState, newF);
}
```

Dengan desain antrean state ini dan penggunaan rekam jejak jalur memori closed list dengan format currentId + '|' + unvisitedIds.join(","), algoritma secara optimal dan sistematis akan memutus eksplorasi rute duplikat. Hasil output dari fungsi ini adalah serangkaian array himpunan urutan objek laporan berurut mulai dari lokasi titik start (Petugas) hingga rute penghentian terakhir.

B. *Hasil Pengembangan*

Dari hasil kalkulasi aplikasi, algoritma A* berhasil mereduksi kombinasi jalur faktorial yang berpotensi menjadi satu untaian lintasan terurut.

Sistem menyajikan luaran dalam dua format, yaitu daftar urutan kunjungan dan visualisasi pada map.



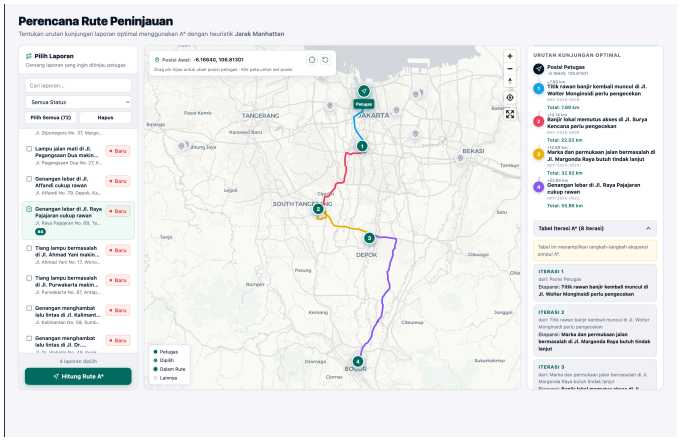
Gambar 1. Tampilan page routing yang menunjukan hasil rute terpendek

Untuk memverifikasi kebenaran langkah algoritma A* dan mengonfirmasi pencarian nilai f(n) terkecil pada setiap langkahnya, aplikasi turut mencatat riwayat ekspansi simpul (node tracing). Tabel berikut memperlihatkan log iterasi kalkulasi antrean prioritas dari sistem saat memecahkan skenario pengujian, yang menampilkan status simpul hidup (open list), simpul yang dipilih untuk diekspansi (best node), beserta nilai kalkulasi g(n), h(n), dan f(n) masing-masing kandidat.

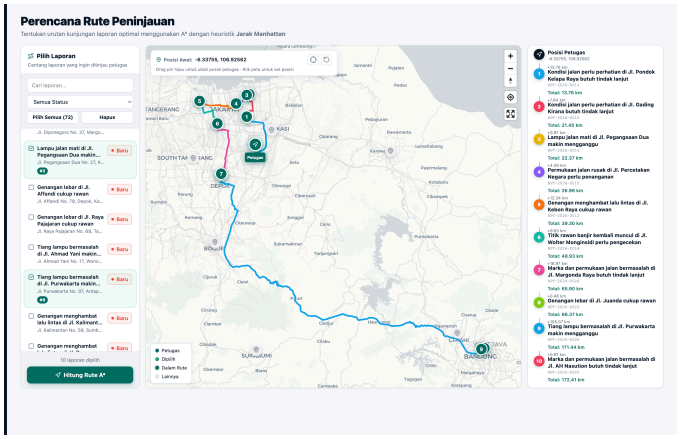
Gambar 2. Tampilan detail hasil kalkulasi

C. *Pengujian Kasus*

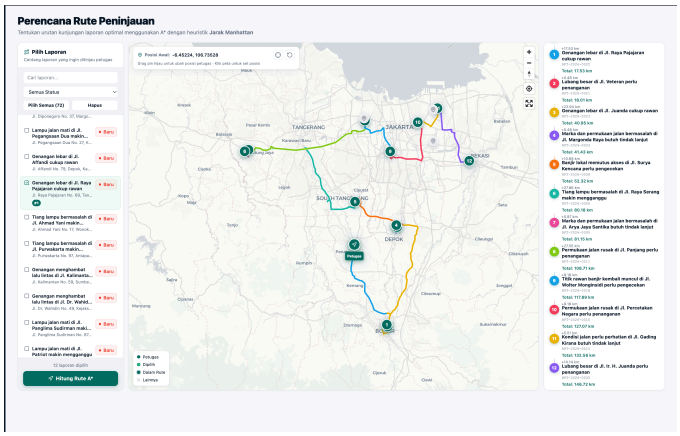
Sistem akan diuji dengan beberapa skenario, pertama adalah skenario linier (Petugas berada di Jakarta, laporan berurut searah ke arah Bogor), kedua adalah skenario menyebar (Laporan menumpuk di Jakarta dan Depok, namun ada 2 laporan di Bandung), dan yang terakhir adalah skenario TSP (12 laporan acak). Hasilnya adalah sebagai berikut:



Gambar 3. Tampilan skenario pertama



Gambar 4. Tampilan skenario kedua



Gambar 5. Tampilan skenario ketiga

Hasil menunjukan bahwa pada skenario pertama rute bergerak lurus tanpa adanya bolak-balik. Lalu pada skenario kedua juga terlihat bahwa rute dilakukan dengan menyelesaikan seluruh hal di satu titik dahulu (Jakarta dan Depok) baru bergerak ke Bandung. Terakhir, pada skenario ketiga menunjukan bahwa program telah menyelesaikan pencarian rute dengan baik.

D. Analisis Hasil

Dari beberapa skenario yang telah dilakukan, terlihat bahwa pada setiap iterasi, simpul hidup (open list) dihitung nilai g(n) akumulatifnya dari posisi awal. Kemudian sisa laporan dievaluasi menggunakan Manhattan distance terjauh untuk menyumbang h(n). Lalu simpul dengan f(n) = h(n) + g(n) terendah akan dimasukkan ke dalam daftar simpul terekspansi (closed list) dan ditandai dengan sub-rute selanjutnya.

Apabila kita membandingkannya dengan Greedy Best-First Search yang sering kali menemui jebakan optimum lokal karena hanya mengejar h(n) minimal [1], A* membuktikan bahwa dengan menyertakan beban masa lalu yaitu g(n), tidak akan terjadi rute terjebak yang membutuhkan petugas berbalik mundur dari ujung kota ke tempat yang mungkin sekitarnya sudah pernah dikunjungi olehnya.

Mengingat pencarian rute ini bertujuan untuk menyapu seluruh titik laporan, persoalan ini setara dengan Traveling Salesperson Problem (TSP) yang merupakan persoalan NP-Hard. Secara teoretis, ruang pencarian (state space) maksimal dari algoritma A* untuk TSP dibatasi oleh jumlah kombinasi titik saat ini dan sub-himpunan simpul yang belum dikunjungi. Jika terdapat N buah titik laporan, maka jumlah maksimum status yang unik adalah $N * 2^N$. Pada setiap ekspansi simpul, algoritma melakukan iterasi sebanyak maksimal N untuk mencari tetangga, lalu menambahkan simpul ke dalam Priority Queue yang membutuhkan waktu $O(\log(N2^N)) = O(N)$. Oleh karena itu, kompleksitas waktu asimptotik terburuk (worst-case) dari implementasi algoritma ini adalah $O(N^2*2^N)$.

Sedangkan untuk kompleksitas ruang (space complexity), algoritma membutuhkan ruang memori untuk menyimpan open list (di dalam Priority Queue) dan closed list (di dalam struktur Map hash table). Dalam kasus terburuk, program harus menyimpan seluruh status yang mungkin di memori, sehingga kompleksitas ruangnya adalah $O(N*2^N)$.

E. Dampak Penggunaan Heuristik terhadap Pemangkasan Cabang Pencarian (Pruning)

Untuk membuktikan efisiensi komputasional dari algoritma A* yang diimplementasikan, kita dapat membandingkannya secara teoretis dengan algoritma pencarian buta (uninformed search) seperti Uniform Cost Search (UCS). Jika sistem menggunakan UCS, algoritma akan mengekspansi rute secara radial ke segala arah berdasarkan nilai g(n) semata. Pada skenario laporan yang menyebar (misalnya ada laporan di Jakarta dan Bandung), UCS akan mencoba mengeksplorasi ribuan kombinasi sub-path di sekitar Jakarta tanpa memiliki "insting" untuk segera bergerak ke Bandung, sehingga antrean open list akan membesar secara eksponensial dan membebani memori (heap out of memory).Dengan masuknya Jarak Manhattan sebagai h(n), evaluasi f(n) = g(n) + h(n) secara dinamis menarik arah pencarian (seperti medan magnet) ke arah titik yang secara geometris berada di jalur yang benar. Cabang-cabang rute (state) yang membawa petugas semakin menjauh

dari sisa kelompok titik laporan akan memiliki nilai h(n) yang melonjak tajam. Akibatnya, state tersebut akan tenggelam di dasar Min-Heap Priority Queue dan tidak akan diekspansi kecuali benar-benar diperlukan. Hal ini secara drastis menekan jumlah simpul yang harus dibangkitkan, membuat memori Node.js pada lingkungan backend tetap stabil meskipun memproses hingga belasan titik laporan warga.

V. KESIMPULAN DAN SARAN

A. *Kesimpulan*

Penerapan Algoritma A* menggunakan kombinasi fungsi matematis Haversine sebagai ukuran bobot g(n) dan Manhattan Distance sebagai ukuran estimasi heuristik h(n) merupakan solusi strategis, cepat, dan presisi untuk menuntaskan masalah routing dalam operasional pelayanan publik.

Implementasi ini berhasil dikemas dalam bentuk perencana rute interaktif yang dapat mensimulasikan ekspansi pohon secara langsung, lengkap dengan proyeksi pemetaan jalan raya geometris. Algoritma A* secara praktis membuktikan kapasitasnya sebagai pencarian heuristik komprehensif yang dijamin optimal sesuai teori komputasinya.

B. *Saran*

Meskipun implementasi algoritma A* pada sistem CommunityMap telah berjalan, namun terdapat beberapa batasan dan asumsi penyederhanaan yang dilakukan pada pemodelan ini. Ke depannya, penelitian dan pengembangan sistem ini dapat disempurnakan melalui beberapa aspek berikut:

1) *Perhitungan Jarak Jalan Nyata (Real Road Network)*

Saat ini, komponen biaya aktual g(n) dihitung menggunakan Haversine Distance (jarak lurus lengkung bumi) dan h(n) menggunakan Manhattan Distance. Pada dunia nyata, petugas lapangan terhalang oleh topologi infrastruktur seperti gedung, jalan satu arah, dan sungai. Pengembangan selanjutnya dapat mengintegrasikan API Routing eksternal seperti Google Maps Distance Matrix API atau OpenStreetMap Routing Machine (OSRM) untuk mendapatkan nilai bobot sisi graf (waktu tempuh atau metrik jalan) yang faktual secara real-time.

2) *Penerapan Algoritma Lain untuk Skala Besar*

Karena algoritma A* menyelesaikan masalah TSP guna mencari solusi eksak (absolute optimal), kompleksitasnya berada pada domain permasalahan NP-Hard dengan laju $O(N^2*2^N)$. Algoritma ini sangat responsif untuk jumlah titik laporan di bawah 15-20 titik. Namun, jika dalam satu hari terdapat ratusan laporan infrastruktur jalan rusak berskala masif, metode ini akan mengalami kendala combinatorial explosion. Oleh karena itu, pada skala makro, penggunaan algoritma lain yang lebih efisien sangat dibutuhkan.

REFERENCES

[1] N. U. Maulidevi, "Penentuan Rute (Route/Path Planning) Bagian 1: BFS, DFS, UCS, Greedy Best First Search," Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika ITB, Bandung, 2026.

[2] N. U. Maulidevi dan Rinaldi M., "Penentuan Rute (Route/Path Planning) Bagian 2: Algoritma A*," Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika ITB, Bandung, 2026.

[3] C. Veness, "Calculate distance, bearing and more between Latitude/Longitude points," Movable Type Scripts. [Online]. Available: https://www.movable-type.co.uk/scripts/latlong.html.

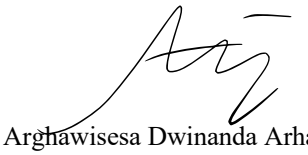
LAMPIRAN

Github: https://github.com/arghaarham/CommunityMap
Video: https://youtu.be/Gy3vffMmpU8

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.

Kota Tangerang, 19 Juni 2026



Arghawisesa Dwinanda Arham
13524100