Implementasi Algoritma UCS, GBFS, dan A* pada Sistem Travel Planner untuk Optimasi Pemilihan Hotel dan Rute Wisata

Reynard Anderson Wijaya - 13524111
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: reynard150806@gmail.com , 13524111@std.stei.itb.ac.id

***Abstract*—Pemilihan hotel dan rute wisata menjadi masalah yang sering dialami wisatawan karena berbagai pertimbangan yang perlu diperhitungkan seperti jarak dan keterbatasan anggaran penginapan. Maka, makalah ini bertujuan untuk mencari rute wisata dan penginapan yang paling optimal dengan mengimplementasikan tiga algoritma pencarian: *Uniform Cost Search* (UCS), *Greedy Best-First Search* (GBFS), dan A*. Sistem Travel Planner yang dibuat akan menghasilkan output berupa tabel ranking penginapan dan rute wisata berdasarkan destinasi dan anggaran yang ditentukan. Hasil implementasi menunjukkan bahwa UCS dan A* dapat memberikan hasil rute yang optimal, sedangkan GBFS lebih cepat dalam mencari rute wisata.**

Keywords—uniform cost search; greedy best-first search; A* search; travel planner; optimasi rute; rekomendasi hotel

I. Pendahuluan

Perencanaan perjalanan wisata tidak hanya perlu memilih destinasi yang ingin dikunjungi, tetapi juga memilih penginapan yang sesuai dengan kebutuhan dan anggaran wisatawan. Semakin banyak destinasi yang ingin dikunjungi, semakin sulit untuk menentukan hotel yang strategis dan rute perjalanan yang efisien. Oleh karena itu, diperlukan suatu metode yang dapat membantu wisatawan memperoleh rekomendasi penginapan dan rute perjalanan yang optimal.

Permasalahan tersebut dapat dimodelkan sebagai graf berbobot, di mana destinasi wisata dan penginapan direpresentasikan sebagai node, sedangkan jarak antar lokasi direpresentasikan sebagai edge. Dengan model ini, pencarian rute dapat dilakukan menggunakan algoritma pencarian yang umum digunakan dalam kecerdasan buatan.

Pada makalah ini diimplementasikan tiga algoritma pencarian, yaitu *Uniform Cost Search* (UCS), *Greedy Best-First Search* (GBFS), dan A*. UCS melakukan pencarian berdasarkan biaya total terkecil sehingga dapat menghasilkan solusi optimal. GBFS memanfaatkan informasi heuristik untuk mempercepat proses pencarian, meskipun solusi yang diperoleh tidak selalu optimal. Sementara itu, A* menggabungkan biaya aktual dan nilai heuristik sehingga dapat menghasilkan solusi optimal dengan proses pencarian yang lebih efisien.

Ketiga algoritma tersebut diterapkan pada sistem Travel Planner berbasis Java yang memungkinkan pengguna memilih beberapa destinasi wisata serta menentukan batas anggaran penginapan per malam. Sistem kemudian memberikan rekomendasi hotel dan menampilkan rute perjalanan yang dihasilkan oleh masing-masing algoritma. Melalui implementasi ini, dilakukan perbandingan terhadap karakteristik UCS, GBFS, dan A* dalam menyelesaikan permasalahan pemilihan hotel dan optimasi rute wisata.

II. Landasan Teori

A. *Uniform Cost Search (UCS)*

Uniform Cost Search adalah jenis algoritma pencarian yang membandingkan setiap edge dengan bobot yang berbeda. Algoritma ini menggunakan struktur data *priority queue* untuk menentukan urutan eksplorasi node, di mana node dengan biaya total $g(n)$ terkecil akan dieksplorasi terlebih dahulu.

Pada setiap iterasi, node dengan biaya terkecil dikeluarkan dulu dari *priority queue*, dan diperiksa apakah node tersebut merupakan node tujuan atau bukan. Kalau bukan, seluruh node tetangga yang belum dikunjungi akan ditambahkan ke dalam *queue*. Proses ini berulang sampai node tujuan ditemukan atau seluruh node telah dieksplorasi (Seperti BFS).

UCS menjamin ditemukannya solusi dengan total biaya minimum, karena algoritma akan selalu mengeksplorasi node dengan biaya paling kecil. Sifat ini menjadikan UCS sebagai algoritma yang optimal, namun masih lemah dalam efisiensi komputasi karena tidak memiliki informasi arah node tujuan.

Pseudocode Algoritma UCS adalah sebagai berikut.

```
procedure UCS(start, goal)
    frontier ← priority queue berisi (start, 0)
    visited ← himpunan kosong
    while frontier tidak kosong do
        n ← simpul dengan g(n) terkecil pada frontier
        if n anggota visited then continue
        tambahkan n ke visited
        if n = goal then
            return RECONSTRUCTPATH(n)
        end if
        for each tetangga m dari n do
            if m bukan anggota visited then
                g(m) ← g(n) + bobot(n, m)
```

```
                masukkan (m, g(m)) ke frontier
            end if
        end for
    end while
    return gagal
end procedure
```

B. *Greedy Best-First Search (GBFS)*

Greedy Best-First Search adalah algoritma pencarian yang menggunakan fungsi heuristik $h(n)$ sebagai dasar pengambilan keputusan dalam menentukan node mana yang akan dieksplorasi. Nilai $h(n)$ merepresentasikan estimasi jarak dari node saat ini menuju node tujuan, tanpa memperhitungkan biaya yang telah dikeluarkan.

Pada setiap iterasi, GBFS memilih node dengan nilai $h(n)$ terkecil dari seluruh node yang berada dalam *priority queue* untuk dieksplorasi berikutnya. Pendekatan ini membuat GBFS lebih cepat dalam mencapai solusi, karena algoritma lebih agresif dalam mengarahkan pencarian menuju tujuan tanpa mempertimbangkan jalur alternatif.

Oleh karena itu, GBFS memiliki kelemahan dalam optimalitas solusi karena algoritma ini tidak mempertimbangkan biaya $g(n)$ yang dikeluarkan. Karakteristik tersebut menjadikan GBFS sebagai algoritma yang unggul dalam efisiensi waktu eksplorasi, namun kurang dapat diandalkan apabila optimalitas solusi menjadi prioritas utama.

Pseudocode Algoritma GBFS adalah sebagai berikut.

```
procedure GBFS(start, goal)
    frontier ← priority queue berisi (start, h(start))
    visited ← himpunan kosong
    while frontier tidak kosong do
        n ← simpul dengan h(n) terkecil pada frontier
        if n ∈ visited then continue
        tambahkan n ke visited
        if n = goal then
            return RECONSTRUCTPATH(n)
        end if
        for each tetangga m dari n do
            if m ∉ visited then
                masukkan (m, h(m)) ke frontier
            end if
        end for
    end while
    return gagal
end procedure
```

C. *A* Search*

A* Search adalah algoritma pencarian yang menggabungkan kelebihan UCS dan GBFS, yaitu dengan komponen $g(n)$ dan $h(n)$ ke dalam satu fungsi evaluasi, yaitu $f(n) = g(n) + h(n)$. Dengan mempertimbangkan kedua komponen tersebut secara bersamaan, A* dapat mengarahkan pencarian menuju tujuan secara efisien tanpa mengorbankan biaya yang telah ditempuh.

Syarat utama agar A* dapat menjamin solusi optimal adalah fungsi heuristik yang digunakan harus bersifat *admissible*, yaitu nilai $h(n)$ tidak pernah melebih-lebihkan biaya sebenarnya yang dibutuhkan untuk mencapai node tujuan. Apabila kondisi ini dipenuhi, A* menjamin penemuan solusi dengan biaya total minimum, sekaligus mengeksplorasi jumlah node yang lebih sedikit dibandingkan UCS karena arah pencarian yang lebih efisien.

Pseudocode Algoritma A* adalah sebagai berikut.

```
procedure ASTAR(start, goal)
    frontier ← priority queue berisi (start, f = h(start))
    g(start) ← 0
    while frontier tidak kosong do
        n ← simpul dengan f(n) terkecil pada frontier
        if n = goal then
            return RECONSTRUCTPATH(n)
        end if
        for each tetangga m dari n do
            g' ← g(n) + bobot(n, m)
            if g' < g(m) (atau m belum dikunjungi) then
                g(m) ← g'
                f(m) ← g(m) + h(m)
                masukkan (m, f(m)) ke frontier
            end if
        end for
    end while
    return gagal
end procedure
```

D. *Graf Berbobot sebagai Model Representasi*

Graf berbobot digunakan sebagai struktur data utama untuk menunjukkan hubungan antar lokasi dalam sistem Travel Planner. Setiap destinasi wisata dan hotel direpresentasikan sebagai node pada graf, sedangkan jarak tempuh antar lokasi direpresentasikan sebagai edge berbobot. Representasi ini memodelkan permasalahan pemilihan hotel dan rute wisata sebagai permasalahan pencarian jalur terpendek pada graf yang akan diselesaikan menggunakan algoritma pencarian UCS, GBFS, dan A*.

Struktur graf pada sistem ini disimpan dalam bentuk *adjacency list*, yaitu setiap node menyimpan daftar edge yang terhubung dengannya secara langsung. Karena setiap node pada umumnya hanya terhubung dengan beberapa node tetangga saja, *adjacency list* memungkinkan proses pencarian tetangga node pada saat eksplorasi dilakukan secara cepat dan efisien.

III. METODOLOGI

A. *Perumusan Masalah*

Permasalahan rekomendasi hotel dan optimasi rute wisata yang dibahas dapat diselesaikan menggunakan pendekatan perhitungan total jarak tempuh dari setiap penginapan yang tersedia ke seluruh destinasi terpilih menggunakan algoritma pencarian jalur terpendek, kemudian melakukan penyaringan terhadap penginapan yang melebihi anggaran yang ditetapkan. Pendekatan ini memungkinkan seluruh penginapan tetap dipertimbangkan dalam proses pe-ranking-an, namun memprioritaskan hotel yang memenuhi batasan anggaran sebagai rekomendasi utama yang ditampilkan kepada pengguna.

B. *Representasi Data sebagai Graf*

Permasalahan dimodelkan sebagai graf berbobot tidak berarah, di mana setiap destinasi wisata dan hotel direpresentasikan sebagai node, sedangkan jarak tempuh antar lokasi direpresentasikan sebagai edge berbobot non-negatif. Representasi ini memungkinkan permasalahan diselesaikan menggunakan algoritma pencarian UCS, GBFS, dan A*. Pada implementasi sistem, graf dibangun dengan enam belas node destinasi wisata dan delapan node hotel yang saling terhubung melalui edge yang ditentukan secara manual, sehingga graf dapat merepresentasikan kasus di mana jarak tempuh aktual tidak selalu proporsional dengan jarak garis lurus pada peta.

C. *Strategi Perhitungan Jarak Hotel ke Destinasi*

Untuk setiap hotel pada graf, sistem menghitung total jarak tempuh ke seluruh destinasi wisata yang dipilih dengan menjalankan ketiga algoritma pencarian secara terpisah. Hasil pencarian dari ketiga algoritma kemudian dijumlahkan untuk mendapatkan tiga nilai total jarak per hotel.

D. *Strategi Pe-ranking-an dan Penyaringan Hotel*

Hotel di-ranking berdasarkan nilai jarak minimum di antara ketiga total jarak yang diperoleh dari UCS, GBFS, dan A* untuk hotel tersebut. Strategi ini dipilih agar peringkat akhir merepresentasikan rute terbaik yang dapat dicapai oleh suatu hotel. Setelah itu, sistem melakukan penyaringan terhadap harga per malam masing-masing hotel agar tidak melebihi anggaran maksimum yang ditetapkan. Hotel dengan harga yang melebihi anggaran akan diberi label "Di Luar Budget", sedangkan hotel dengan peringkat terbaik yang harganya masih berada dalam batas anggaran akan dipilih sebagai rekomendasi utama. Ketiga jalur algoritma dari hotel rekomendasi tersebut kemudian divisualisasikan pada peta.

IV. IMPLEMENTASI

A. *Representasi Lokasi*

Setiap node pada graf direpresentasikan oleh kelas Location yang menyimpan atribut identitas (id, nama), jenis lokasi (wisata atau hotel), koordinat posisi pada peta, dan atribut tambahan khusus hotel berupa rating dan harga per malam. Kelas ini juga memiliki metode heuristicTo() yang menghitung jarak Euclidean antara dua lokasi berdasarkan koordinatnya (dipakai dalam GBFS dan A*).

```
public double heuristicTo(Location target) {
    double dx = this.x - target.x;
    double dy = this.y - target.y;
    return Math.sqrt(dx * dx + dy * dy) * 10; // skala ke satuan km (asumsi)
}
```

Gambar 4.1. Potongan Kode heuristicTo()

B. *Struktur Graf*

Kelas Graph menyimpan node pada Map<String, Location> dan menyimpan hubungan antar node dalam bentuk adjacency list pada Map<String, List<Edge>>. Setiap edge bersifat dua arah sehingga penambahan satu edge akan menghasilkan dua entri pada *adjacency list*.

```
public void addEdge(String fromId, String toId, double weight) {
    Location from = locations.get(fromId);
    Location to   = locations.get(toId);
    if (from == null || to == null) return;

    Edge e1 = new Edge(from, to, weight);
    Edge e2 = new Edge(to, from, weight);

    edges.add(e1);
    adjacency.get(fromId).add(e1);
    adjacency.get(toId).add(e2);
}
```

Gambar 4.2. Potongan Kode addEdge()

C. *Implementasi Uniform Cost Search*

UCS diimplementasikan menggunakan PriorityQueue yang mengurutkan node berdasarkan biaya total cost ($g(n)$) secara menaik. Pada setiap iterasi, node dengan biaya terkecil dikeluarkan dari queue. Apabila node tersebut belum pernah dikunjungi, node ditandai sebagai dikunjungi dan seluruh tetangganya yang belum dikunjungi dimasukkan kembali ke dalam queue dengan biaya total baru. Proses berhenti ketika node tujuan ditemukan, sehingga jalur dan total biaya yang diperoleh terjamin minimum.

```
public SearchResult ucs(String startId, String goalId) {
    SearchResult result = new SearchResult(algorithm: "UCS");
    long startTime = System.currentTimeMillis();

    PriorityQueue<PQNode> pq = new PriorityQueue<>();
    Map<String, Double> visited = new HashMap<>();

    pq.add(new PQNode(locations.get(startId), cost: 0, priority: 0, parent: null));

    while (!pq.isEmpty()) {
        PQNode current = pq.poll();
        String cid = current.location.getId();

        if (visited.containsKey(cid)) continue;
        visited.put(cid, current.cost);
        result.nodesExplored++;

        if (cid.equals(goalId)) {
            result.path = reconstructPath(current);
            result.totalCost = current.cost;
            break;
        }

        for (Edge edge : adjacency.getOrDefault(cid, new ArrayList<>())) {
            String nid = edge.to.getId();
            if (!visited.containsKey(nid)) {
                double newCost = current.cost + edge.weight;
                pq.add(new PQNode(edge.to, newCost, newCost, current));
            }
        }
    }

    result.timeMs = System.currentTimeMillis() - startTime;
    return result;
}
```

Gambar 4.3. Potongan Kode ucs()

D. *Implementasi Greedy Best-First Search*

GBFS menggunakan PriorityQueue yang sama, namun nilai priority yang digunakan untuk mengurutkan queue hanya berasal dari nilai heuristik $h(n)$, tanpa memperhitungkan biaya total $g(n)$. Hal ini terlihat pada baris pq.add(new PQNode(edge.to, current.cost + edge.weight, h, current)), di mana parameter cost tetap dicatat untuk keperluan pelacakan jalur, tetapi parameter priority yang menentukan urutan eksplorasi hanya diisi oleh nilai h. Karena totalCost berbasis heuristik tidak merepresentasikan jarak aktual, total biaya

akhir dihitung ulang dengan menjumlahkan bobot edge aktual di sepanjang jalur yang ditemukan, alih-alih menggunakan nilai cost pada node tujuan secara langsung.

```java
public SearchResult gbfs(String startId, String goalId) {
    SearchResult result = new SearchResult(algorithm: "GBFS");
    long startTime = System.currentTimeMillis();

    Location goal = locations.get(goalId);
    PriorityQueue<PQNode> pq = new PriorityQueue<>();
    Set<String> visited = new HashSet<>();

    double h0 = locations.get(startId).heuristicTo(goal);
    pq.add(new PQNode(locations.get(startId), cost: 0, h0, parent: null));

    while (!pq.isEmpty()) {
        PQNode current = pq.poll();
        String cid = current.location.getId();

        if (visited.contains(cid)) continue;
        visited.add(cid);
        result.nodesExplored++;

        if (cid.equals(goalId)) {
            result.path = reconstructPath(current);

            result.totalCost = 0;
            List<Location> path = result.path;
            for (int i = 0; i < path.size() - 1; i++) {
                String fid = path.get(i).getId();
                String tid = path.get(i + 1).getId();
                for (Edge e : adjacency.getOrDefault(fid, new ArrayList<>())) {
                    if (e.to.getId().equals(tid)) { result.totalCost += e.weight; break; }
                }
            }
            break;
        }

        for (Edge edge : adjacency.getOrDefault(cid, new ArrayList<>())) {
            String nid = edge.to.getId();
            if (!visited.contains(nid)) {
                double h = edge.to.heuristicTo(goal);

                pq.add(new PQNode(edge.to, current.cost + edge.weight, h, current));
            }
        }
    }

    result.timeMs = System.currentTimeMillis() - startTime;
    return result;
}
```

Gambar 4.4. Potongan Kode gbfs()

E. *Implementasi A* Search*

A* menggunakan fungsi evaluasi $f(n) = g(n) + h(n)$, yang diimplementasikan dengan mempertahankan gScore sebagai map biaya total terbaik yang telah ditemukan untuk setiap node. Setiap kali ditemukan jalur baru dengan tentative yang lebih kecil dibandingkan gScore yang tersimpan sebelumnya, node tersebut diperbarui dan dimasukkan kembali ke dalam queue dengan nilai priority $f(n)$ yang baru.

```java
public SearchResult aStar(String startId, String goalId) {
    SearchResult result = new SearchResult(algorithm: "A*");
    long startTime = System.currentTimeMillis();

    Location goal = locations.get(goalId);
    PriorityQueue<PQNode> pq = new PriorityQueue<>();
    Map<String, Double> gScore = new HashMap<>();

    double h0 = locations.get(startId).heuristicTo(goal);
    pq.add(new PQNode(locations.get(startId), cost: 0, h0, parent: null));
    gScore.put(startId, value: 0.0);

    while (!pq.isEmpty()) {
        PQNode current = pq.poll();
        String cid = current.location.getId();
        result.nodesExplored++;

        if (cid.equals(goalId)) {
            result.path = reconstructPath(current);
            result.totalCost = current.cost;
            break;
        }

        for (Edge edge : adjacency.getOrDefault(cid, new ArrayList<>())) {
            String nid = edge.to.getId();
            double tentativeG = current.cost + edge.weight;

            if (tentativeG < gScore.getOrDefault(nid, Double.MAX_VALUE)) {
                gScore.put(nid, tentativeG);
                double h = edge.to.heuristicTo(goal);
                double f = tentativeG + h;
                pq.add(new PQNode(edge.to, tentativeG, f, current));
            }
        }
    }

    result.timeMs = System.currentTimeMillis() - startTime;
    return result;
}
```

Gambar 4.5. Potongan Kode aStar()

F. *Antarmuka Pengguna*

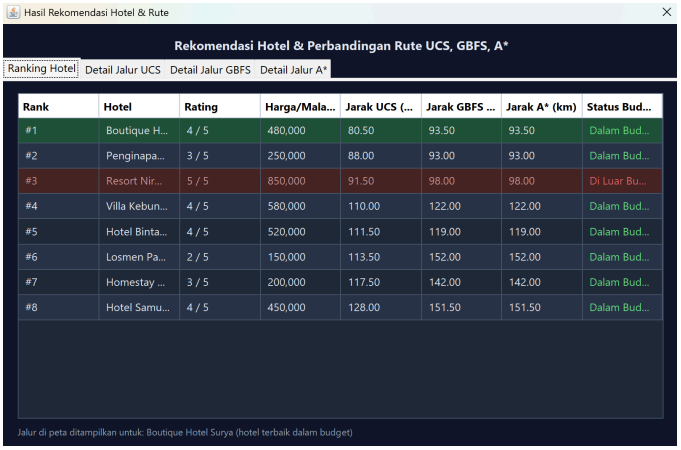
Antarmuka grafis sistem dibangun menggunakan Java Swing, terdiri dari panel pemilihan destinasi wisata berupa checkbox, input anggaran hotel, panel visualisasi peta interaktif yang menampilkan node dan jalur hasil pencarian, serta dialog hasil yang menampilkan tabel peringkat hotel beserta detail jalur dari masing-masing algoritma dalam bentuk tab terpisah. Visualisasi peta membedakan jalur UCS, GBFS, dan A* menggunakan warna serta ketebalan garis yang berbeda untuk memudahkan perbandingan secara visual.

V. Hasil dan Pembahasan

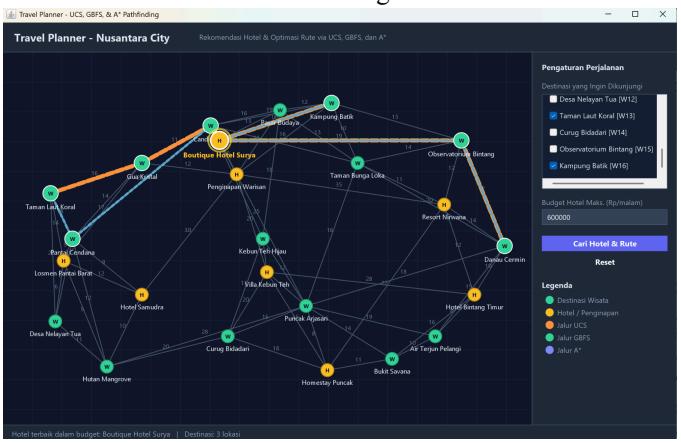
Pengujian sistem dilakukan dengan menjalankan tiga skenario berbeda untuk mengamati perilaku UCS, GBFS, dan A* pada variasi jumlah destinasi wisata dan batas anggaran hotel. Setiap skenario menghasilkan dua bentuk keluaran, yaitu tabel peringkat hotel yang menampilkan total jarak ke seluruh destinasi terpilih menurut ketiga algoritma, serta visualisasi peta yang menunjukkan jalur hasil pencarian dari hotel rekomendasi ke setiap destinasi.

A. *Skenario 1: Tiga Destinasi dengan Budget Rp 600.000*

Pada skenario ini, pengguna memilih tiga destinasi wisata, yaitu Taman Laut Koral, Kampung Batik, dan Observatorium Bintang, dengan batas anggaran hotel sebesar Rp 600.000 per malam.



Gambar 5.1. Tabel Ranking Hotel Skenario 1

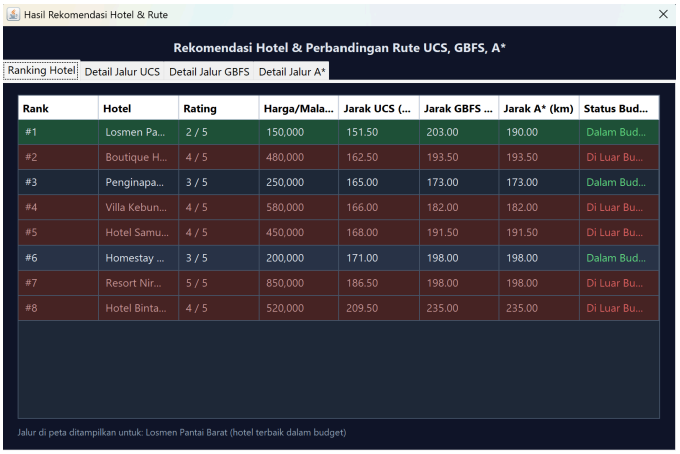


Gambar 5.2. Peta Jalur Skenario 1

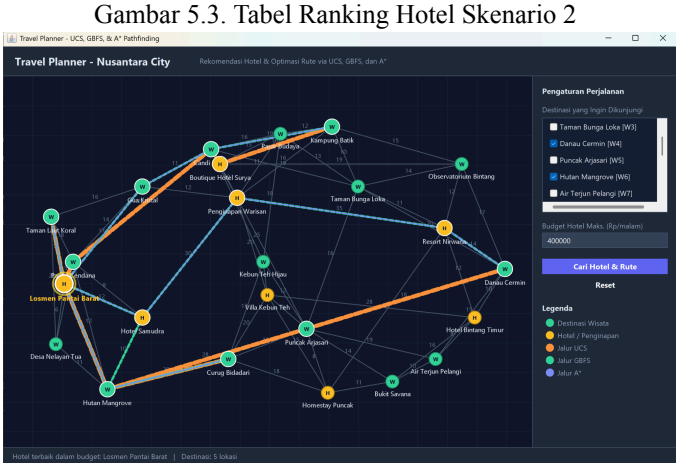
Berdasarkan Gambar 5.1., Boutique Hotel Surya terpilih sebagai rekomendasi utama dengan total jarak UCS sebesar 80,50 km, sedangkan GBFS dan A* menghasilkan total jarak yang identik, yaitu 93,50 km. Resort Nirwana, meskipun memiliki rating tertinggi (5/5), berada pada peringkat ketiga dan berstatus di luar budget karena harganya (Rp 850.000) melebihi batas anggaran yang ditetapkan. Pada skenario ini terlihat bahwa GBFS menghasilkan jarak yang lebih besar dibandingkan UCS pada seluruh hotel yang terdaftar, menunjukkan bahwa pendekatan greedy berbasis heuristik tidak berhasil menemukan jalur optimal pada struktur graf dengan destinasi yang saling berjauhan ini.

B. Skenario 2: Lima Destinasi dengan Budget Rp 400.000

Pada skenario ini, pengguna memilih lima destinasi wisata yang lebih tersebar, yaitu Danau Cermin, Hutan Mangrove, dan tiga destinasi lain pada rentang anggaran yang lebih rendah, yaitu Rp 400.000 per malam.



Gambar 5.3. Tabel Ranking Hotel Skenario 2

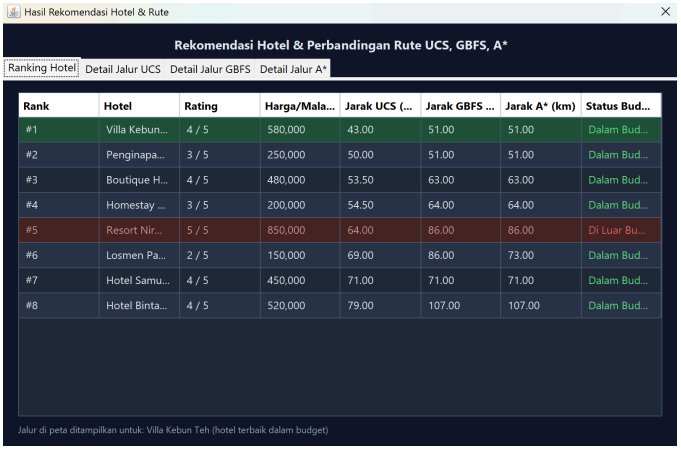


Gambar 5.4. Peta Jalur Skenario 2

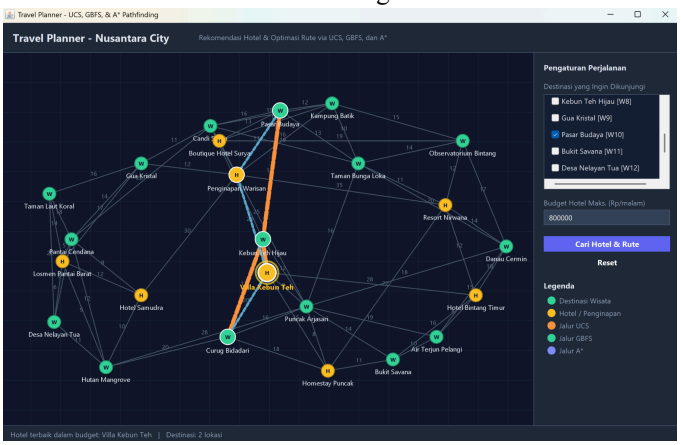
Hasil pada Gambar 5.3. menunjukkan Losmen Pantai Barat sebagai hotel dengan total jarak UCS terkecil (151,50 km) sekaligus berada dalam batas anggaran. Menariknya, pada skenario ini A* (190,00 km) menghasilkan jarak yang lebih kecil dibandingkan GBFS (203,00 km) untuk hotel yang sama, meskipun keduanya menggunakan fungsi heuristik yang sama. Hal ini terjadi karena A* tetap mempertimbangkan biaya aktual $g(n)$ dalam pengambilan keputusan, sehingga dapat menghindari jalur yang secara heuristik terlihat menjanjikan namun pada akhirnya membutuhkan jarak tempuh lebih jauh, sedangkan GBFS murni mengikuti estimasi heuristik tanpa koreksi tersebut. Pada visualisasi peta Gambar 5.4., perbedaan ini tampak jelas melalui jalur GBFS (garis hijau) yang mengambil rute berbeda dengan jalur A* (garis ungu) pada beberapa segmen, khususnya pada lintasan menuju Hutan Mangrove.

C. Skenario 3: Dua Destinasi dengan Budget Rp 800.000

Pada skenario ketiga, pengguna memilih dua destinasi wisata, yaitu Pasar Budaya, dengan batas anggaran yang lebih luas sebesar Rp 800.000 per malam.



Gambar 5.5. Tabel Ranking Hotel Skenario 3



Gambar 5.6. Peta Jalur Skenario 3

Pada skenario dengan jumlah destinasi paling sedikit ini, Villa Kebun Teh terpilih sebagai rekomendasi utama dengan total jarak UCS sebesar 43,00 km, sementara GBFS dan A* kembali menghasilkan nilai yang identik, yaitu 51,00 km. Konsistensi antara GBFS dan A* pada skenario ini, berbeda dengan Skenario 2, mengindikasikan bahwa pada graf dengan jumlah destinasi yang lebih sedikit dan jarak antar node yang relatif berdekatan, heuristik jarak Euclidean cukup akurat dalam mengarahkan pencarian tanpa terjebak pada jalur lokal yang menyesatkan. Visualisasi pada Gambar 6 juga memperlihatkan bahwa jalur UCS (oranye) dan jalur GBFS/A* (hijau-ungu yang saling bertumpuk) mengambil rute yang hampir serupa namun tidak identik, terlihat dari sedikit perbedaan lintasan di sekitar node Kebun Teh Hijau.

VI. KESIMPULAN

Berdasarkan implementasi dan pengujian yang telah dilakukan terhadap tiga skenario dengan variasi jumlah destinasi dan batas anggaran, dapat disimpulkan bahwa ketiga algoritma yang diimplementasikan, yaitu UCS, GBFS, dan A*, memiliki karakteristik yang saling melengkapi namun tidak setara dalam hal optimalitas solusi.

UCS secara konsisten menghasilkan total jarak terkecil pada seluruh skenario uji, sesuai dengan sifat teoritisnya yang menjamin solusi optimal tanpa bergantung pada kualitas fungsi heuristik. Namun, jaminan optimalitas ini diperoleh dengan mengeksplorasi node secara lebih merata ke segala arah, sehingga UCS menjadi algoritma yang paling dapat diandalkan dari segi akurasi, tetapi tidak selalu paling efisien dari segi komputasi pada graf yang lebih kompleks.

A* terbukti sebagai algoritma yang paling seimbang di antara ketiganya. Pada Skenario 1 dan 3, A* menghasilkan total jarak yang identik dengan GBFS, namun pada Skenario 2 yang melibatkan jumlah destinasi terbanyak, A* berhasil mendekati performa UCS (190,00 km berbanding 151,50 km secara total leg, dengan selisih yang jauh lebih kecil dibandingkan GBFS) berkat mekanisme pembaruan $g(n)$ yang memungkinkan algoritma mengoreksi keputusan berdasarkan biaya aktual. Karakteristik ini menjadikan A* sebagai strategi yang paling optimal untuk diterapkan secara umum pada sistem Travel Planner, karena memberikan kombinasi terbaik antara akurasi solusi dan efisiensi eksplorasi node, tanpa mengorbankan kualitas rute secara signifikan dibandingkan UCS.

GBFS, di sisi lain, hanya optimal digunakan pada kasus dengan struktur graf yang sederhana dan jumlah destinasi sedikit, seperti yang terlihat pada Skenario 1 dan 3. Pada Skenario 2 dengan kompleksitas lebih tinggi, GBFS menghasilkan total jarak yang signifikan lebih besar dibandingkan A* maupun UCS, sehingga algoritma ini kurang disarankan sebagai strategi utama apabila pengguna mengunjungi banyak destinasi sekaligus, meskipun keunggulannya dalam kecepatan eksplorasi tetap menjadikannya pilihan yang relevan untuk kebutuhan pencarian cepat tanpa mempertimbangkan optimalitas mutlak.

Secara keseluruhan, sistem Travel Planner yang dikembangkan berhasil mendemonstrasikan bahwa pemilihan algoritma pencarian memiliki dampak nyata terhadap kualitas rekomendasi rute wisata, dan bahwa strategi mengambil nilai jarak minimum di antara ketiga algoritma untuk keperluan perankingan hotel merupakan pendekatan yang tepat untuk memastikan rekomendasi tetap merepresentasikan performa terbaik yang dapat dicapai, terlepas dari algoritma mana yang menghasilkannya. Untuk pengembangan lebih lanjut, sistem ini dapat diperluas dengan mempertimbangkan kriteria tambahan seperti rating hotel dan preferensi waktu kunjungan, serta penerapan pada graf dengan skala yang lebih besar untuk menguji konsistensi temuan ini pada kondisi dunia nyata yang lebih kompleks.

LAMPIRAN

Github:
https://github.com/RND815/Travel-Planner-with-UCS-GBFS-A-Algorithm

Video:
https://drive.google.com/file/d/1xuPLVIXC4yBhUUPj_F2QsKAIGwSm2TOY/view?usp=sharing

UCAPAN TERIMA KASIH

Penulis mengucapkan syukur kepada Tuhan Yang Maha Esa atas kesabaran dan kesempatan yang diberikan sehingga makalah ini dapat diselesaikan dengan tepat waktu. Penulis juga ingin berterima kasih kepada Ir. Rinaldi Munir, M.T., dosen mata kuliah IF2211 Strategi Algoritma, atas bimbingan

yang diberikan selama masa perkuliahan. Penulis juga berterima kasih kepada teman-teman, keluarga, dan semua pihak lain yang telah memberikan dukungan dan semangat selama penyusunan makalah ini.

References

[1] R. Munir, "Route Planning (Bagian 1)," 2026. [Online]. Available: http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/21-Route-Planning-(2026)-Bagian1.pdf. [Diakses: 11-Juni-2026].

[2] R. Munir, "Route Planning (Bagian 2)," 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/22-Route-Planning-(2026)-Bagian2.pdf. [Diakses: 11-Juni-2026].

[3] V. Chugani, "What is Manhattan Distance? A Deep Dive," DataCamp, 2024. [Online]. Available: https://www.datacamp.com/tutorial/manhattan-distance. [Diakses: 11-Juni-2026].

[4] Oracle, "JFrame (Java Platform SE 8)," Java Platform Standard Edition 8 Documentation. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html. [Diakses: 13-Juni-2026].

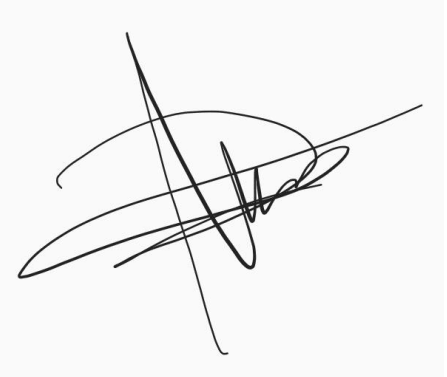
[5] Oracle, "Creating a GUI With JFC/Swing," The Java Tutorials. [Online]. Available: https://docs.oracle.com/javase/tutorial/uiswing/. [Diakses: 13-Juni-2026].

[6] Oracle, "How to Use Swing Components," The Java Tutorials. [Online]. Available: https://docs.oracle.com/javase/tutorial/uiswing/components/. [Diakses: 13-Juni-2026].

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, 17 Juni 2026



Reynard Anderson Wijaya
13524111