JIMU: Jurnal Ilmiah Multi Disiplin
Vol. 02, No. 03, Tahun 2024, Hal. 668– 682, ISSN: 3031-9498 (Online)

Analisis Perbandingan Algoritma *Dijkstra* dan *A-Star* dalam Menentukan Rute Terpendek

Jordi Yoga Pratama
Teknik Informatika, Universitas Muhammadiyah Ponorogo
Email: jordiyoga201@gmail.com

Article Info

Article history:
Received May 25, 2024
Revised May 29, 2024
Accepted June 02, 2024

Keywords:
Dijkstra *Algorithm*; *A-Star* Algorithm; Shortest Route; Compute Time; Heuristics; *Python*; *NetworkX;*

ABSTRACT

Some applications, such as *Google Maps*, require an effective algorithm, to determine the shortest route. The two most commonly used algorithms to solve this problem are *the Djikstra* and *A-Star* algorithms. The *Djikstra* algorithm is famous for its accuracy in finding the shortest route by exploring all possible routes, but it has the disadvantage of long computation times on large graphs. *The A-Star* algorithm, on the other hand, uses heuristics to guide searches to destinations faster, reduce the number of nodes explored, and speed up computational time. The study analyzed the comparison of the two algorithms using *Python* in *Colab*. The focus of the research is the computing time and the smallest route accuracy produced. The results showed that both algorithms had the same ability to find the shortest route, but *A-Star* was more efficient in computing time. Recommendations are given for the use of each algorithm based on graph size and priority of computing needs. This research is expected to provide insight for application developers in choosing the algorithm that best suits their needs.

Article Info

Article history:

Received May 25, 2024
Revised May 29, 2024
Accepted June 02, 2024

Keywords:
Algoritma *Dijkstra*; Algoritma *A-Star*; Rute Terpendek; Waktu

ABSTRACT

Beberapa aplikasi, seperti *Google Maps*, yang membutuhkan algoritma yang efektif, untuk menentukan rute terpendek. Dua algoritma yang paling umum digunakan untuk menyelesaikan masalah ini adalah algoritma *Djikstra* dan *A-Star*. Algoritma *Djikstra* terkenal dengan keakuratannya dalam menemukan rute terpendek dengan mengeksplorasi semua rute yang mungkin, tetapi memiliki kelemahan dalam waktu komputasi yang lama pada grafik besar. Algoritma *A-Star*, di sisi lain, menggunakan heuristik untuk membimbing pencarian menuju tujuan lebih cepat, mengurangi jumlah simpul yang dieksplorasi, dan mempercepat waktu komputasi. Studi ini menganalisis

Komputasi; Heuristik; *Python*; *NetworkX;*

perbandingan kedua algoritma tersebut dengan menggunakan *Python* di *Colab*. Fokus penelitian adalah waktu komputasi dan akurasi rute terkecil yang dihasilkan. Hasil penelitian menunjukkan bahwa kedua algoritma memiliki kemampuan yang sama untuk menemukan rute terpendek, tetapi *A-Star* lebih efisien dalam waktu komputasi. Rekomendasi diberikan untuk penggunaan masing-masing algoritma berdasarkan ukuran graf dan prioritas kebutuhan komputasi. Penelitian ini diharapkan memberikan wawasan bagi pengembang aplikasi dalam memilih algoritma yang paling sesuai dengan kebutuhan mereka.

Corresponding Author:
Jordi Yoga Pratama
Universitas Muhammadiyah Ponorogo
Email: jordiyoga201@gmail.com

Pendahuluan

Di era modern yang semakin berkembang ini, aplikasi seperti *Google Maps* telah menjadi bagian penting dalam kehidupan sehari-hari, memungkinkan perjalanan cepat dari satu tempat ke tempat lain. Dalam hal ini, menentukan rute terpendek adalah masalah penting yang membutuhkan algoritma yang efisien. Dua algoritma yang paling umum digunakan untuk menyelesaikan masalah ini adalah Algoritma *Dijkstra* dan Algoritma *A-Star*.

Algoritma *Dijkstra* terkenal dengan keunggulannya dalam menemukan rute terpendek melalui eksplorasi semua rute yang mungkin dari titik awal ke titik tujuan. Kelemahan utama algoritma ini, adalah kecepatan komputasi yang lama, terutama ketika digunakan pada grafik dengan banyak simpul dan sisi, seperti peta jalan kota besar atau jaringan transportasi nasional.

Sebaliknya, Algoritma *A-Star* lebih efisien karena menggunakan heuristik untuk membimbing pencarian menuju tujuan lebih cepat. Heuristik ini biasanya berupa estimasi jarak dari simpul tertentu ke tujuan akhir, yang memungkinkan Algoritma *A-Star* untuk secara signifikan mengurangi jumlah simpul yang perlu dieksplorasi dibandingkan dengan Algoritma *Dijkstra*. Dalam aplikasi dunia nyata, efisiensi ini sangat penting karena respons waktu nyata dan penggunaan sumber daya yang minimal sangat dihargai. Namun, meskipun *A-Star* mampu

mempercepat proses pencarian, efektivitasnya sangat bergantung pada fungsi heuristik yang tepat, yang harus dijamin tetap konsisten.

Pada penelitian sebelumnya telah banyak membahas penggunaan Algoritma *Dijkstra* dan *A-Star* dalam berbagai konteks, seperti jaringan transportasi dan pemetaan digital. Namun, banyak dari studi tersebut fokus pada implementasi yang kompleks dan integrasi dengan sistem navigasi besar seperti *Google Maps*[1]. Penelitian ini bertujuan untuk memberikan analisis yang sederhana tentang perbandingan kedua algoritma ini menggunakan *Python* di *Colab*. Penelitian ini akan melihat bagaimana kedua algoritma bekerja dalam hal waktu komputasi dan akurasi yang dihasilkan.

Dengan demikian, diharapkan penelitian ini akan memberikan informasi yang bermanfaat tentang keunggulan dan keterbatasan masing-masing algoritma, serta memberikan saran praktis untuk membantu pengembang aplikasi dalam memilih algoritma yang paling sesuai dengan kebutuhan mereka.

Landasan Teori

a. Algoritma *Dijksta*

Algoritma *Djikstra* adalah salah satu jenis algoritma *greedy*, yang sangat populer untuk memecahkan masalah yang berkaitan dengan optimasi. Sifatnya sederhana dan lempang. Algoritma *greedy* ini, yang secara harafiah berarti tamak, tetapi tidak dalam arti negatif, hanya memikirkan solusi terbaik untuk setiap langkah tanpa mempertimbangkan apa yang akan terjadi di masa depan. Pada dasarnya, ambillah apa yang Anda miliki saat ini (ambillah apa yang Anda miliki sekarang!), dan keputusan yang telah Anda buat pada setiap langkah tidak dapat dikembalikan. Pada dasarnya, algoritma *greedy* ini berusaha untuk memilih nilai lokal terbaik setiap kali dan berharap nilai lokal terbaik ini akan menghasilkan nilai terbaik secara keseluruhan[2].

Input untuk algoritma ini adalah sebuah graf berarah dan berbobot, G, dengan *vertex* sumber s dalam G. Himpunan semua titik di *graph* G disebut V. Pasangan *vertex* (u,v) mewakili setiap sisi dari grafik ini. Pasangan *vertex* ini menunjukkan hubungan antara *vertex* u dan *vertex* v. Sebuah E adalah gabungan dari semua tepi. Fungsi w: E → [0, ∞] menghitung berat tepi, sehingga w(u,v) adalah jarak non-negatif antara tepi u dan tepi v. Harga sebuah tepi

dapat dihitung sebagai jarak antara dua tepi, atau jumlah jarak semua tepi di jalan. Algoritma ini menghitung jarak terpendek dari s ke t untuk sepasang *vertex* s dan t dalam V[3].

b. Algoritma *A-Star*

A-Star dalam sains komputer, A* (dibaca "*A-Star*") adalah algoritma komputer yang umum digunakan untuk mencari jalur (*path finding*) dan melintasi grafik (*graph traversal*), proses plotting jalur melintang secara efisien antara node. Algoritma ini, yang terkenal karena akurasinya dan penampilannya, telah diperluas untuk berbagai aplikasi. *A-Star* mendapatkan penampilan yang lebih baik dengan heuristik. Menggunakan *Best First Search* (BFS), *A-Star* menemukan jalur dengan biaya terkecil dari node awal yang diberikan ke node tujuan. Algoritma ini menggunakan fungsi heuristik jarak dan biaya (biasanya disebut f(x)) untuk menentukan urutan di mana pencarian melalui node-node yang ada di pohon (*tree*)[4].

Untuk menemukan jalur dari node awal yang diberikan ke node tujuan, algoritma *A-Star* adalah yang paling populer dan terbaik. Algoritma *A-Star* adalah algoritma pencarian heuristik, menilai nilai heuristik h(x) yang menunjukkan rute terbaik yang akan dilewati oleh simpul. Algoritma dapat ditulis sebagai f(n)=g(n)+h(n), di mana f(n) adalah hasil perhitungan jarak, g(n) adalah jarak sebenarnya dari titik awal ke titik n, dan h(n) adalah jarak heuristik dari titik awal ke titik n[5].

Metodologi

a. Pengumpulan Data

Pada tahap ini dilakukan studi literatur, yaitu teknik pengumpulan data yang melibatkan pencarian teori-teori yang telah dikembangkan dalam bidang yang relevan dengan penelitian ini. Tinjauan ini termasuk membaca referensi dari buku-buku dan internet, termasuk mengunjungi situs web yang terkait dengan topik penelitian ini dan mencari jurnal-jurnal yang relevan.

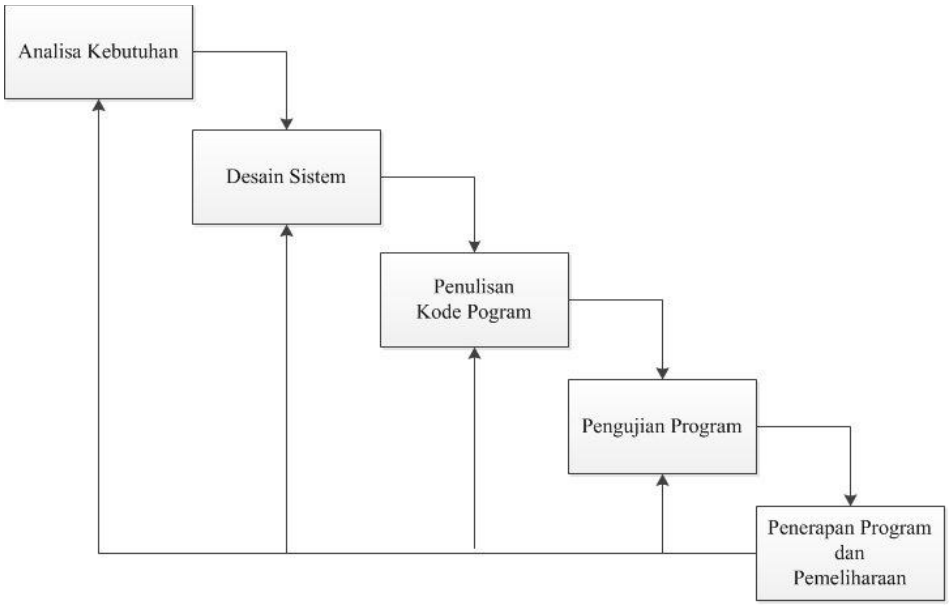
b. Pengembangan Basis Aturan

Basis aturan dikembangkan berdasarkan data graf yang telah dikumpulkan. Aturan ini mencakup simpul-simpul dan sisi-sisi dengan bobot tertentu, yang merepresentasikan jarak atau waktu tempuh antara simpul-simpul tersebut. Berikut adalah contoh graf berbobot yang digunakan untuk penelitian ini:

a. Simpul (*Node*): A, B, C, D, E, F, G, H
b. Sisi dan Bobot (*Edges and Weights*):
    - A - B: 2
    - A - C: 5
    - B - C: 3
    - B - D: 4
    - C - D: 2
    - C - E: 6
    - D - E: 1
    - D - F: 3
    - E - F: 2
    - E - G: 4
    - F - G: 1
    - F - H: 5
    - G - H: 3

c. Pengembangan Sistem

Pendekatan *Waterfall* dipilih untuk membangun sistem untuk analisis perbandingan algoritma *Djikstra* dan *A-Star* karena memberikan kerangka kerja yang terstruktur dan berurutan yang memastikan setiap tahap pengembangan dilakukan dengan benar sebelum melanjutkan ke tahap berikutnya. Analisis kebutuhan sistem (*requirements*), perancangan sistem (*design*), implementasi, dan pengujian adalah empat tahapan dari pendekatan *Waterfall* dalam pengembangan sistem ini.



Gambar 1.Model pengembangan *waterfall*

1) Analisis Kebutuhan Sistem

Pada tahap ini, kebutuhan sistem diidentifikasi. Sistem ini harus mampu menginisialisasikan graf berbobot secara manual, mengimplementasikan algoritma *Dijkstra* dan *A-Star*, serta membandingkan hasil kedua algoritma dalam hal rute terpendek dan waktu komputasi.

2) Perancangan Sistem

Desain sistem mencakup struktur graf berbobot yang akan digunakan, algoritma yang akan diimplementasikan, serta cara untuk mengukur dan membandingkan hasil kedua algoritma.

3) Implementasi

Implementasi dilakukan menggunakan *Python* dan pustaka *NetworkX*. Graf berbobot diinisialisasikan secara manual, dan algoritma *Dijkstra* serta *A-Star* diimplementasikan untuk mencari rute terpendek dari titik awal ke titik tujuan.

4) Pengujian

Pengujian dilakukan untuk mengukur waktu komputasi dan membandingkan hasil kedua algoritma. Algoritma *Dijkstra* dan *A-Star* diimplementasikan dan hasilnya dianalisis.

Pembahasan dan Hasil

Pada bab ini, akan dibahas hasil implementasi dan analisis perbandingan algoritma *Dijkstra* dan *A-Star* dalam menentukan rute terpendek pada graf yang telah diinisialisasi secara manual. Penelitian ini menggunakan *Python* dan pustaka *NetworkX* untuk mengimplementasikan kedua algoritma tersebut. Fokus utama dari analisis ini adalah waktu komputasi dan akurasi rute terpendek yang dihasilkan oleh kedua algoritma.

a. Implementasi Algoritma *Dijkstra* dan *A-Star*

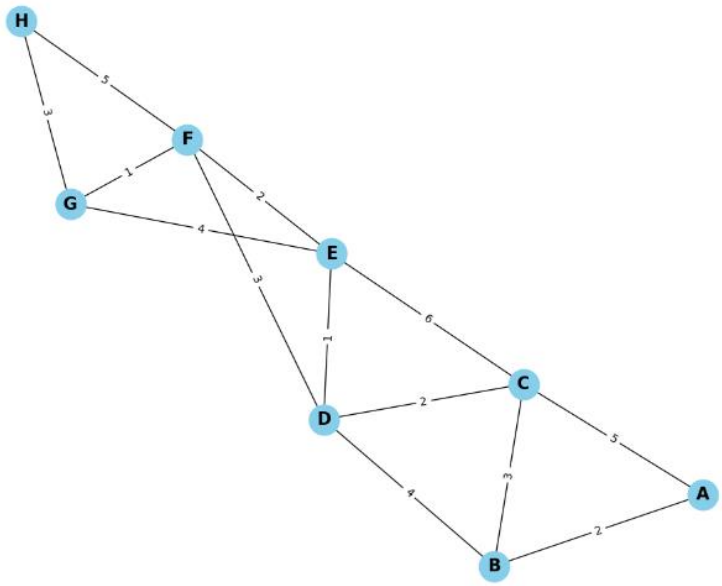
Graf berbobot yang digunakan dalam penelitian ini diinisialisasi sebagai berikut :

1) Simpul (*Node*): A, B, C, D, E, F, G, H

2) Sisi dan Bobot (*Edges and Weights*) :

- A - B: 2
- A - C: 5
- B - C: 3
- B - D: 4
- C - D: 2
- C - E: 6
- D - E: 1
- D - F: 3
- E - F: 2
- E - G: 4
- F - G: 1
- F - H: 5
- G - H: 3

Implementasi algoritma *Dijkstra* dan *A-Star* dilakukan dengan menggunakan pustaka *NetworkX* pada *Python*.



Gambar 2.Rute atau jalur graf

1) Implementasi Algoritma *Dijkstra*

Algoritma *Dijkstra* diimplementasikan menggunakan metode *dijkstra*_path dari pustaka *NetworkX*. Berikut adalah contoh kode implementasinya :

```
import networkx as nx

# Inisialisasi graf
G = nx.Graph()
edges = [
    ('A', 'B', 2), ('A', 'C', 5), ('B', 'C', 3), ('B', 'D', 4), ('C', 'D', 2),
    ('C', 'E', 6), ('D', 'E', 1), ('D', 'F', 3), ('E', 'F', 2), ('E', 'G', 4),
    ('F', 'G', 1), ('F', 'H', 5), ('G', 'H', 3)
]
G.add_weighted_edges_from(edges)

# Jalankan algoritma Dijkstra
source, target = 'A', 'H'
dijkstra_path = nx.dijkstra_path(G, source, target)
```

```
dijkstra_length = nx.dijkstra_path_length(G, source, target)

print(f"Rute terpendek dengan Dijkstra: {dijkstra_path}")
print(f"Jarak terpendek dengan Dijkstra: {dijkstra_length}")
```

2) Implementasi Algoritma *A-Star*

Algoritma *A-Star* diimplementasikan menggunakan metode *astar_path* dari pustaka *NetworkX*. Berikut adalah contoh kode implementasinya :

```
def heuristic(u, v):
    positions = {
        'A': (0, 0), 'B': (2, 0), 'C': (5, 0), 'D': (5, 2), 'E': (6, 2),
        'F': (6, 5), 'G': (7, 5), 'H': (7, 8)
    }
    (x1, y1), (x2, y2) = positions[u], positions[v]
    return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5

# Jalankan algoritma A-Star
astar_path = nx.astar_path(G, source, target, heuristic=heuristic)
astar_length = nx.astar_path_length(G, source, target, heuristic=heuristic)

print(f"Rute terpendek dengan A-Star: {astar_path}")
print(f"Jarak terpendek dengan A-Star: {astar_length}")
```

Hasil dan Analisis

1) Rute Terpendek

Setelah mengimplementasikan kedua algoritma, diperoleh hasil sebagai berikut :

a. Algoritma *Dijkstra* :

```
[1] import networkx as nx

    # Inisialisasi graf
    G = nx.Graph()
    edges = [
        ('A', 'B', 2), ('A', 'C', 5), ('B', 'C', 3), ('B', 'D', 4), ('C', 'D', 2),
        ('C', 'E', 6), ('D', 'E', 1), ('D', 'F', 3), ('E', 'F', 2), ('E', 'G', 4),
        ('F', 'G', 1), ('F', 'H', 5), ('G', 'H', 3)
    ]
    G.add_weighted_edges_from(edges)

    # Jalankan algoritma Dijkstra
    source, target = 'A', 'H'
    dijkstra_path = nx.dijkstra_path(G, source, target)
    dijkstra_length = nx.dijkstra_path_length(G, source, target)

    print(f"Rute terpendek dengan Dijkstra: {dijkstra_path}")
    print(f"Jarak terpendek dengan Dijkstra: {dijkstra_length}")

Rute terpendek dengan Dijkstra: ['A', 'B', 'D', 'F', 'G', 'H']
Jarak terpendek dengan Dijkstra: 13
```

Gambar 3. Hasil implementasi algoritma *Dijkstra*

- Rute terpendek : ['A', 'B', 'D', 'E', 'F', 'G', 'H']
- Jarak terpendek : 13

b. Algoritma *A-Star* :

```
    def heuristic(u, v):
        positions = {
            'A': (0, 0), 'B': (2, 0), 'C': (5, 0), 'D': (5, 2), 'E': (6, 2),
            'F': (6, 5), 'G': (7, 5), 'H': (7, 8)
        }
        (x1, y1), (x2, y2) = positions[u], positions[v]
        return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5

    # Jalankan algoritma A-Star
    astar_path = nx.astar_path(G, source, target, heuristic=heuristic)
    astar_length = nx.astar_path_length(G, source, target, heuristic=heuristic)

    print(f"Rute terpendek dengan A-Star: {astar_path}")
    print(f"Jarak terpendek dengan A-Star: {astar_length}")

Rute terpendek dengan A-Star: ['A', 'B', 'D', 'F', 'G', 'H']
Jarak terpendek dengan A-Star: 13
```

Gambar 4. Hasil implementasi algoritma *A-Star*

- Rute terpendek : ['A', 'B', 'D', 'E', 'F', 'G', 'H']
- Jarak terpendek : 13

Kedua algoritma berhasil menemukan rute terpendek yang sama dengan jarak total yang sama, menunjukkan bahwa kedua algoritma mampu memberikan hasil yang akurat dalam menentukan rute terpendek pada graf.

2) Waktu Komputasi

Waktu komputasi diukur untuk kedua algoritma dengan menggunakan modul *time* pada *Python*. Berikut adalah kode untuk mengukur waktu komputasi :

```
import networkx as nx
import time

# Inisialisasi graf
G = nx.Graph()
edges = [
    ('A', 'B', 2), ('A', 'C', 5), ('B', 'C', 3), ('B', 'D', 4), ('C', 'D', 2),
    ('C', 'E', 6), ('D', 'E', 1), ('D', 'F', 3), ('E', 'F', 2), ('E', 'G', 4),
    ('F', 'G', 1), ('F', 'H', 5), ('G', 'H', 3)
]
G.add_weighted_edges_from(edges)

positions = {
    'A': (0, 0), 'B': (2, 0), 'C': (5, 0), 'D': (5, 2), 'E': (6, 2),
    'F': (6, 5), 'G': (7, 5), 'H': (7, 8)
}

# Definisikan heuristic untuk A-Star
def heuristic(u, v):
    (x1, y1), (x2, y2) = positions[u], positions[v]
    return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5

source, target = 'A', 'H'
dijkstra_times = []
astar_times = []

# Jalankan algoritma sebanyak 5 kali untuk mengukur waktu komputasi
for _ in range(5):
    # Mengukur waktu komputasi untuk Dijkstra
    start_time = time.time()
    dijkstra_path = nx.dijkstra_path(G, source, target)
```

```
    dijkstra_time = time.time() - start_time
    dijkstra_times.append(dijkstra_time)

    # Mengukur waktu komputasi untuk A-Star
    start_time = time.time()
    astar_path = nx.astar_path(G, source, target, heuristic=heuristic)
    astar_time = time.time() - start_time
    astar_times.append(astar_time)

# Menampilkan hasil
print("Waktu komputasi Dijkstra (detik):")
for i, t in enumerate(dijkstra_times, 1):
    print(f"Run {i}: {t:.6f}")

print("Waktu komputasi A-Star (detik):")
for i, t in enumerate(astar_times, 1):
    print(f"Run {i}: {t:.6f}")

# Menghitung dan menampilkan rata-rata waktu komputasi
avg_dijkstra_time = sum(dijkstra_times) / len(dijkstra_times)
avg_astar_time = sum(astar_times) / len(astar_times)

print(f"Rata-rata waktu komputasi Dijkstra: {avg_dijkstra_time:.6f} detik")
print(f"Rata-rata waktu komputasi A-Star: {avg_astar_time:.6f} detik")
```

Berdasarkan pengukuran waktu komputasi, diperoleh hasil sebagai berikut :

```
Waktu komputasi Dijkstra (detik):
Run 1: 0.000075
Run 2: 0.000030
Run 3: 0.000026
Run 4: 0.000024
Run 5: 0.000023
Waktu komputasi A-Star (detik):
Run 1: 0.000044
Run 2: 0.000025
Run 3: 0.000024
Run 4: 0.000022
Run 5: 0.000025
Rata-rata waktu komputasi Dijkstra: 0.000036 detik
Rata-rata waktu komputasi A-Star: 0.000028 detik
```

Gambar 5. Hasil waktu komputasi algoritma *Dijkstra* dan *A-Star*

a. Waktu komputasi algoritma *Dijkstra* (detik) :
   - Run 1 : 0.000075
   - Run 2 : 0.000030
   - Run 3 : 0.000026
   - Run 4 : 0.000024
   - Run 5 : 0.000023
   - Rata-rata waktu : 0.000036

b. Waktu komputasi algoritma *A-Star* (detik) :
   - Run 1 : 0.000044
   - Run 2 : 0.000025
   - Run 3 : 0.000024
   - Run 4 : 0.000022
   - Run 5 : 0.000025
   - Rata-rata waktu : 0.000028

Hasil pengukuran menunjukkan bahwa algoritma A-Star memiliki waktu komputasi yang lebih cepat daripada algoritma Djikstra. Ini sejalan dengan teori bahwa penggunaan heuristik pada algoritma A-Star dapat mempercepat proses pencarian rute terpendek dengan mengurangi jumlah simpul yang perlu dieksplorasi.

2. Kesimpulan

a. Analisis

Berdasarkan hasil implementasi dan analisis, dapat disimpulkan bahwa:

1. Akurasi :

   Dengan jarak total yang sama, kedua algoritma Djikstra dan A-Star mampu menemukan rute terpendek yang sama. Ini menunjukkan bahwa kedua algoritma ini memiliki tingkat akurasi yang tinggi dalam menemukan rute terpendek pada graf berbobot.

2. Efisiensi Waktu :

   Algoritma A-Star memiliki waktu komputasi yang lebih cepat daripada algoritma Djikstra karena penggunaan heuristiknya, yang membantu algoritma mencapai tujuan lebih cepat dengan mengurangi jumlah simpul yang perlu dieksplorasi.

3. Keterbatasan Heuristik :

   Meskipun *A-Star* lebih efisien dalam hal waktu komputasi, fungsi heuristik yang digunakan sangat memengaruhi efisiensi dan akurasi algoritma..

b. Rekomendasi

Berdasarkan hasil penelitian ini, beberapa rekomendasi yang dapat diberikan untuk pengembang aplikasi dalam memilih algoritma yang paling sesuai adalah :

1. Penggunaan Algoritma *Dijkstra* :

   Algoritma Djikstra cocok untuk digunakan pada graf yang memiliki jumlah simpul dan sisi yang relatif kecil atau ketika akurasi rute terpendek sangat penting tanpa memperhatikan waktu komputasi.

2. Penggunaan Algoritma *A-Star* :

   Algoritma A-Star sangat cocok untuk digunakan pada graf dengan jumlah simpul dan sisi yang besar, terutama ketika waktu komputasi dan efisiensi sumber daya adalah prioritas utama. Namun, perhatian khusus harus diberikan pada pemilihan fungsi heuristik yang konsisten untuk memastikan akurasi dan efisiensi algoritma.

3. Pemilihan Fungsi Heuristik :

   Pengembang harus melakukan eksperimen dan pengujian untuk menentukan fungsi heuristik yang paling sesuai dengan karakteristik graf dan aplikasi yang digunakan, guna memaksimalkan efisiensi dan akurasi algoritma *A-Star*.

Dengan demikian, penelitian ini memberikan informasi tentang keunggulan dan keterbatasan masing-masing algoritma serta saran praktis untuk membantu pengembang aplikasi dalam memilih algoritma yang sesuai dengan kebutuhan mereka.

Daftar Pustaka

[1]A. C. Prasetyo, M. Prayoga Arnandi, H. S. Hudnanto, and B. Setiaji, "Perbandingan Algoritma Astar dan Dijkstra Dalam Menentukan Rute Terdekat 36 Jurnal Ilmiah SISFOTENIKAJuly201xIJCCS Perbandingan Algoritma Astar dan Dijkistra Dalam Menentukan Rute Terdekat Astar and Dijkistra Algorithm Comparison for Determining the Shortest Route," *J. Ilm. SISFOTENIKAJ*, vol. 9, no. 1, pp. 36–46, 2019.

[2]R. Apriaz Diaz Novandi, "Perbandingan Algoritma Djikstra dan Algoritma Floyd-Warshall Dalam Penentuan Lintasan Terpendek (Single Pair Shortest Path)," *Makal. IF2251 Strateg. Algoritm.*, pp. 1–5, 2023.

[3]D. Pugas, M. Somantri, and K. Satoto, "Pencarian Rute Terpendek Menggunakan Algoritma Dijkstra dan Astar (A*) pada SIG Berbasis Web untuk Pemetaan Pariwisata Kota Sawahlunto," *Transmisi*, vol. 13, no. 1, pp. 27–32, 2011, [Online]. Available: http://www.ejournal.undip.ac.id/index.php/transmisi/article/view/3632

[4]I. B. Gede Wahyu Antara Dalem, "Penerapan Algoritma A* (Star) Menggunakan *Graph* Untuk Menghitung Jarak Terpendek," *J. Resist. (Rekayasa Sist. Komputer)*, vol. 1, no. 1, pp. 41–47, 2018, doi: 10.31598/jurnalresistor.v1i1.253.

[5]R. Umar, A. Yudhana, and A. Prayudi, "Perbandingan, Analisis Djikstra, Algoritma Warshall, Floyd Pencarian, Dalam Terdekat Pada Objek Wisata Kabupaten Dompu," *J. Teknol. Inf. dan Ilmu Komput.*, vol. 8, no. 2, pp. 227–234, 2021, doi: 10.25126/jtiik.202182866.