SMASH: Journal of Social Management Sains and Health
E-ISSN: 3089-252X P-ISSN: 3089-2538
Vol. 3 No. 1 April 2026

Analisis Keterbatasan Algoritma Dijkstra dalam Menentukan Jalur Terpendek pada Graph

Ananda Syafika[1] Febrianta Immanuel Purba[2] M Farel Revansha Buulolo[3] Nayla Luthfiah Hanan[4] Adidtya Perdana[5]
Program Studi Ilmu Komputer, Universitas Negeri Medan, Kabupaten Deli Serdang, Provinsi Sumatera Utara, Indonesia[1,2,3,4,5]
Email: anandasyafika402@gmail.com[1] febriantapurba445@gmail.com[2] farelrevansha@gmail.com[3] naylaluthfiahh@gmail.com[4] adidtya@unimed.ac.id[5]

Abstrak

Penelitian ini bertujuan untuk menganalisis keterbatasan algoritma Dijkstra dalam menentukan jalur terpendek pada graph berbobot. Metode penelitian yang digunakan adalah studi literatur dengan pendekatan deskriptif-analisis yang berfokus pada analisis teoritis terhadap prinsip kerja algoritma, kompleksitas komputasi, serta kondisi-kondisi yang mempengaruhi keoptimalan solusi. Hasil kajian menunjukkan bahwa algoritma Djikstra bekerja optimal pada graph statis dengan bobot sisi non-negatif dan memenuhi sifat greedy choice serta optimal substucture. Namun, algoritma ini memiliki beberapa keterbatasan utama, yaitu tidak mampu menangani bobot negatif, tidak dapat mendeteksi negative cycle, serta mengalami penurunan efisiensi pada graph berskala besar dan dinamis. Selain itu, keputusan greedy yang bersifat final menyebabkan algoritma tidak dapat merevisi solusi ketika ditemukan jalur yang lebih pendek pada kondisi tertentu. Oleh karena itu, pemilihan algoritma shortest path harus disesuaikan dengan karakteristik graph dan kebutuhan komputasi. Penelitian ini memberikan pemahaman konseptual mengenai batasan operasional algoritma Djikstra sebagai dasar pemilihan metode yang tepat dalam penyelesaian masalah jalur terpendek.
Kata Kunci: Algoritma Dijkstra, Shorstest Path, Graph Berbobot, Kompleksitas Algoritma, Greedy Algorithm

Abstract

This study aims to analyze the limitations of Djikstra's algorithm in determining the shortest path in weighted graphs. The research method employed is a literature study using a descriptive-analytical approach that focuses on theoritical analysis of the algorithm's working principles, computational complexity, and conditions affecting solution optimally. The findings indicate that Dijkstra's algorithm perfoms optimally on static graphs with non-negative edge weights and satisfies the greedy choice property and optimal substructure. However, the algorithm has several major limitations, including its inability to handle negative weights, inability to detect negative cycles, and decreased efficiency in large-scale and dynamic graphs. Futhermore, the greedy decision-making process is final and cannot be revised when shorter paths are discovered under certain conditions. Therefore, the selection of shortest path algorithms must be adjusted to graph characteristics and computational requirements. This study provides a conceptual understanding of the operational limitations of Djikstra's algorithm as a basis for selecting appropriate methods in shortest path problem solving.
Keywords: *Dijkstra's Algorithm, Shortest Path, Weighted Graph, Algorithm Complexity, Greedy Algorithm*

PENDAHULUAN

Teori graf merupakan salah satu cabang fundamental dalam ilmu komputer dan matematika diskrit yang memiliki peran penting dalam berbagai bidang komputasi modern. Representasi struktur data dalam bentuk graph memungkinkan pemodelan berbagai sistem kompleks, seperti jaringan transportasi, jaringan komputer, sistem logistik, hingga analisis jaringan sosial. Salah satu permasalahan klasik dan paling banyak dikaji dalam teori graf adalah

permasalahan *shortest path* atau pencarian jalur terpendek. Permasalahan *shortest path* bertujuan untuk menentukan jalur dengan total bobot minimum dari suatu simpul sumber menuju simpul tujuan dalam suatu graph berbobot. Secara umum, permasalahan ini dibagi menjadi *single source shortest path* dan *all pairs shortest path*. Dalam konteks single source shortest path, salah satu algoritme yang paling populer dan banyak digunakan adalah Algoritma Dijkstra. Algoritma Dijkstra diperkenalkan oleh Edsger W. Dijkstra pada tahun 1959 dan dikenal sebagai algoritma berbasis pendekatan *greedy.* Algoritma ini bekerja dengan memilih simpul dengan jarak minimum secara iteratif dan memperbarui jarak ke simpul-simpul tetangganya hingga seluruh simpul telah diproses. Dengan kompleksitas waktu yang relatif efisien, terutama ketika diimplementasikan menggunakan struktur data *priority queue,* algoritma ini menjadi solusi utama dalam berbagai aplikasi nyata, seperti sistem navigasi digital dan optimasi jaringan. Namun demikian, meskipun memiliki performa yang baik dalam kondisi tertentu, algoritma Dijkstra tidak bersifat universal. Salah satu keterbatasan mendasar dari algoritma ini adalah ketidakmampuannya menangani graph dengan bobot sisi negatif. Selain itu, pada graph berskala besar dengan jumlah simpul dan sisi yang sangat banyak, efisiensi algoritma dapat menurun tergantung pada struktur data yang digunakan. Dalam konteks graph dinamis, di mana bobot sisi dapat berubah secara real-time, algoritma Djikstra juga memerlukan perhitungan ulang secara menyeluruh, sehingga kurang optimal dibandingkan pendekatan lain.

Seiring berkembangnya penelitian dalam bidang desain dan analisis algoritma, berbagai algoritma alternatif seperti Bellman-Ford, Floyd-Warshall, dan pendekatan heuristik seperti A* telah dikembangkan untuk mengatasi keterbatasan tersebut. Meskipun demikian, kajian yang secara khusus dan komprehensif menganalisis keterbatasan algoritma Djikstra dari sudut pandang teoritis dan kompleksitas masih relatif terbatas dalam bentuk sintesis terstruktur. Berdasarkan latar belakang tersebut, penelitian ini difokuskan pada analisis mendalam terhadap keterbatasan algoritma Dijkstra dalam menentukan jalur terpendek pada graph. Penelitian ini tidak hanya membahas cara kerja algoritma, tetapi juga mengevaluasi batasan matematis, karakteristik kompleksitas, serta kondisi-kondisi di mana algoritma ini tidak memberikan solusi optimal. Dengan demikian, penelitian ini diharapkan dapat memberikan pemahaman yang lebih komprehensif dalam pemilihan algoritma yang tepat sesuai dengan karakteristik permasalahan.

Kajian Literatur

Konsep Dasar Graph

Graph merupakan struktur data yang digunakan untuk merepresentasikan hubungan antar objek dalam bentuk simpul (*vertex*) dan sisi (*edge*). Secara matematis, graph dinyatakan sebagai pasangan himpunan $G = (V, E)$, di mana V merupakan himpunan simpul dan E merupakan himpunan sisi yang menghubungkan simpul-simpul tersebut. Graph banyak digunakan dalam berbagai bidang seperti jaringan komputer, transportasi, dan sistem navigasi. Berdasarkan arah sisinya, graph dibagi menjadi dua jenis, yaitu *directed graph* (graf berarah) dan *undirected graph* (graf tak berarah). Pada graf berarah, setiap sisi memiliki arah tertentu, sedangkan pada graf tak berarah hubungn antar simpul bersifat dua arah. Selain itu, graph juga dapat berupa *weighted graph*, yaitu graph yang setiap sisinya memiliki bobot tertentu. Bobot tersebut dapat merepresentasikan jarak, waktu, atau biaya. Dalam permasalahan pencarian jalur terpendek, graph berbobot menjadi representasi utama karena nilai bobot digunakan untuk menentukan jalur optimal. Dalam implementasi algoritma, graph dapat direpresentasikan dalam beberapa bentuk, yaitu:

1. *Adjacency Matrix*, yaitu representasi graph dalam bentuk matrik berukuran $V \times V$.
2. *Adjacenct List*, yaitu representasi graph menggunakan struktur list untuk menyimpan hubungan antar simpul.

Representasi graph yang digunakan akan mempengaruhi efisiensi algoritma pencarian jalur terpendek yang diterapkan.

Permasalahan Shortest Path

Permasalahan *shortest path* merupakan salah satu masalah fundamental dalam teori graph yang bertujuan untuk menentukan jalur dengan total bobot minimum antara simpul pada suatu graph. Permasalahan *shortest path* memiliki beberapa kategori, salah satunya adalah *Single Source Shortest Path* (*SSSP*), yaitu menentukan jarak terpendek dari satu simpul sumber ke seluruh simpul lainnya. Permasalahan ini banyak digunakan dalam system navigasi, routing jaringan, dan optimasi distribusi. Berdasarkan hasil kajian literatur pada beberapa penelitian, algoritma shortest path diklasifikasikan menjadi algoritma klasik, heuristic, dan hybrid. Algoritma klasik seperti Dijkstra, Bellman-Ford, dan Floyd-Warshall banyak digunakan karena memiliki dasar matematis yang kuat dan jaminan Solusi optimal pada kondisi tertentu. Kompleksitas penyelesaian shortest path dipengaruhi oleh jumlah simpul dan sisi pada graph, serta karakteristik bobot seperti adanya bobot negative atau perubahan bobot secara dinamis.

Algoritma Dijkstra

Algoritma Dijkstra merupakan algoritma yang digunakan untuk menentukan jalur terpendek pada graph berbobot non-negatif. Algoritma ini menggunakan pendekatan greedy, yaitu memilih simpul dengan jarak minimum sementara pada setiap iterasi. Berdasarkan studi literatur, algoritma Dijkstra digunakan dalam berbagai bidang seperti system transportasi dan jaringan computer karena mampu menghasilkan solusi optimal pada graph statis dengan bobot non-negatif. Langkah-langkah algoritma Dijkstra adalah sebagai berikut:

1. Menentukan simpul sumber.
2. Memberikan nilai jarak awal 0 pada simpul sumber dan tak hingga pada simpul lainnya.
3. Memilih simpul dengan jarak minimum.
4. Melakukan relaksasi terhadap simpul tetangga.
5. Mengulangi proses hingga semua simpul diproses.

Pseudocode algoritma Dijkstra:

```
for setiap vertex v:
 dist[v] = tak hingga
dist[source] = 0

selama masih ada vertex yang belum diproses:
 pilih vertex u dengan jarak minimum
 untuk setiap tetangga v:
 jika dist[u] + w(u,v) < dist[v]:
 dist[v] = dist[u] + w(u,v)
```

Kompleksitas waktu algoritma Dijkstra bergantung pada struktur data yang digunakan:

- Menggunakan *adjacency matrix*: $O(V^2)$
- Menggunakan *adjacency list* dan *priority queue*: $O((V + E) log\ V)$

Beberapa penelitian menunjukkan bahwa algoritma Dijkstra memiliki kinerja yang baik pada graph statis, tetapi kurang optimal pada graph dinamis karena tidak dapat menyesuaikan perubahan bobot secara *real-time*.

Algoritma Pembanding

Dalam penelitian shortest path, algoritma Dijkstra sering dibandingkan dengan algoritma lain untuk mengetahui kelebihan dan keterbatasannya.

1. Algoritma Bellman-Ford. Algoritma Bellman-Ford digunakan untuk mencari jalur terpendek pada graph yang memungkinkan adanya bobot negatif. Algoritma ini bekerja dengan melakukan proses relaksasi sebanyak $V - 1$ kali. Kelebihan utama algoritma Bellman-Ford adalah kemampuannya dalam mendeteksi *negative cycle*, namun memiliki kompleksitas waktu lebih besar dibandingkan Dijkstra. Berdasarkan studi komparatif beberapa penelitian, Bellman-Ford lebih fleksibel dibandingkan Dijkstra, tetapi kurang efisien dalam graph berukuran besar.
2. Algoritma Floyd-Warshall. Algoritma Floyd-Warshall digunakan untuk menentukan jalur terpendek antara semua pasangan simpul (*all-pairs shortest path*). Algoritma ini menggunakan pendekatan *dynamic programming* dengan kompleksitas waktu: $O(V^3)$

Penelitian ini menunjukkan bahwa Floyd-Warshall mampu menemukan semua kemungkinan jalur sebelum menentukan jalur optimal, sehingga sering digunakan pada graph dengan jumlah simpul kecil hingga menengah.

Perbedaan Karakteristik Algoritma

Berdasarkan hasil studi literatur, setiap algoritma shortest path memiliki karakteristik berbeda sesuai dengan jenis masalah yang diselesaikan.

Algoritma	Jenis Masalah	Bobot Negatif	Kompleksitas
Dijkstra	Single Source	Tidak	$O((V+E)log\ V)$
Bellman-Ford	Single Source	Ya	$O(VE)$
Floyd-Warshall	All Pairs	Ya	$O(V^3)$

Beberapa penelitian menunjukkan bahwa tidak ada algoritma yang selalu paling optimal untuk semua kondisi, sehingga pemilihan algoritma harus disesuaikan dengan karakteristik graph yang digunakan.

METODE PENELITIAN

Jenis dan Desain Penelitian

Penelitian ini merupakan penelitian deskriptif-analitis dengan pendekatan studi literatur (*library research*) yang bersifat kualitatif-konseptual. Penelitian deskriptif bertujuan menggambarkan secara sistematis karakteristik algoritma Dijkstra, sedangkan pendekatan analitis digunakan untuk mengevaluasi keoptimalan dan keterbatasannya berdasarkan kerangka teori algoritma graf. Objek penelitian adalah algoritma Dijkstra yang diperkenalkan oleh Edsger W.Dijkstra pada tahun 1959 untuk menyelesaikan permasalahan *Single source path* pada graf berbobot non-negatif. Desain penelitian bersifat eksploratif-teoritis karena tidak dilakukan eksperimen empiris, melainkan analisis formal terhadap struktur matematis dan sifat algoritmik.

Kerangka Konseptual Penelitian

Penelitian ini didasarkan pada model graf berbobot yang direpresentasikan sebagai:

$$G = (V, E, w)$$

Keterangan:

- V: himpunan simpul (vertex)
- E: himpunan sisi (edge)
- w: $E \rightarrow$ R: fungsi bobot

Masalah yang dianalisis adalah pencarian jarak minimum dari satu simpul sumber $s \in V$ ke seluruh simpul lain dalam graf. Kerangka analisis mencakup:
1. Validitas greedy choice porperty
2. Optimal substructure
3. Monotonicity property
4. Kompleksitas waktu dan ruang

Pendekatan Penelitian

Pendekatan ini menggunakan empat pendekatan utama:
1. Pendekatan Teoritis. Pendekatan ini dilakukan melalui analisis pembuktian formal terhadap: Invariant algoritma; Proses relaksasi sisi; Kondisi finalisasi simpul; Pendekatan ini bertujuan mengidentifikasi syarat perlu dan cukup agar algoritma Dijkstra menghasilkan solusi optimal.
2. Pendekatan Analisis Kompleksitas. Analisis dilakukan terhadap kompleksitas waktu dan ruang berdasarkan variasi struktur data yaitu: Adjacency matrix → $O(V^2)$; Adjacency list + binary heap → $O((V + E) \log V)$; Fibonacci heap → $O(E + V \log V)$. Evaluasi dilakukan pada dua kondisi graf: Dense graph (E mendekati V^2); Sparse graph (E mendekati V)
3. Pendekatan Komparatif. Dilakukan perbandingan dengan algoritma Bellman-Ford dalam aspek: Kemampuan menangani bobot negatif; Deteksi negative cycle; Stabilitas solusi; Kompleksitas komputasi. Pendekatan ini digunakan untuk menegaskan batasan operasional algoritma Dijkstra.
4. Pendekatan Simulasi Konseptual. Dilakukan simulasi langkah demi langkah pada graf uji sederhana untuk: Mengubah perubahan nilai jarak; Mengidentifikasi kondisi kegagalan pada bobot negatif; Mengevaluasi validitas keputusan greedy. Simulasi bersifat konseptual dan tidak melibatkan perangkat lunak eksperimental.

Selain menggunakan pendekatan teoritis dan analisis kompleksitas, penelitian ini juga dilengkapi dengan implementasi algoritma Dijkstra menggunakan bahasa pemrograman Python sebagai bentuk validasi empiris sederhana. Implementasi dilakukan untuk mensimulasikan proses pencarian jalur terpendek pada graph berbobot non-negatif sesuai dengan studi kasus yang telah dianalisis secara manual. Pendekatan implementatif ini bertujuan untuk:
1. Memverifikasi kesesuaian antara hasil perhitungan manual dengan hasil komputasi algoritma.
2. Mengamati proses kerja algoritma secara lebih konkret melalui eksekusi program.
3. Memberikan gambaran praktis mengenai penerapan algoritma Dijkstra dalam menyelesaikan permasalahan shortest path.

Dengan demikian, penelitian ini tidak hanya bersifat konseptual, tetapi juga didukung oleh simulasi komputasional sederhana sebagai bentuk penguatan analisis.

Sumber Data dan Literatur

Data yang digunakan bersifat sekunder dan diperoleh dari: Buku teks algoritma dan struktur data; Artikel jurnal ilmiah mengenai shortest path problem; Publikasi asli Dijkstra (1959); Literatur pembanding algoritma graf. Sumber dipilih berdasarkan relevansi teoritis dan kredibilitas akademik.

Teknik Analisis Data

Analisis dilakukan melalui tahapan berikut: Identifikasi asumsi formal algoritma; Analisis matematis proses relaksasi; Evaluasi kompleksitas komputasi; Uji konseptual pada graf berbobot negatif; Analisis komparatif terhadap algoritma pembanding.

Validitas dan Batasan Penelitian

Validitas penelitian dijamin melalui: Konsistensi logis pembuktian matematis; Kesesuaian dengan teori graf dan analisis algoritma; Referensi akademik yang kredibel. Batasan penelitian ini meliputi: Tidak dilakukan eksperimen berbasis data real – world; Tidak dianalisis performa berbasis implementasi perangkat lunak; Tidak mencakup varian algoritma modifikasi Dijkstra

Alur Penelitian

Alur penelitian dilakukan sebagai berikut: Identifikasi masalah shortest path; Formulasi model graf berbobot; Analisis prinsip kerja algoritma Dijkstra; Evaluasi kompleksitas waktu dan ruang; Simulasi konseptual graf uji; Identifikasi keterbatasan dan Penarikan kesimpulan teoritis.

HASIL PENELITIAN DAN PEMBAHASAN

Analisis Kinerja Algoritma Dijkstra

Algoritma Dijkstra digunakan untuk menyelesaikan permasalahan *single source shortest path* pada graph berbobot non-negatif dengan pendekatan greedy. Pada setiap iterasi, algoritma memilih simpul dengan jarak minimum sementara untuk kemudian dilakukan proses relaksasi terhadap simpul-simpul yang bertetangga dengannya. Secara matematis, jika d(u) merupakan jarak minimum dari simpul sumber ke simpul u, maka proses relaksasi dilakukan dengan persamaan:

$$\boldsymbol{d(v) = \min d(v), d(u) + w((u,v))}$$

Di mana w(u,v) merupakan bobot sisi yang menghubungkan simpul u dan v. Keoptimalan algoritma Dijkstra bergantung pada terpenuhinya *greedy choice property* dan *optimal substructure*. Ketika simpul dengan jarak minimum dipilih, nilai tersebut dianggap sebagai jarak final dan tidak akan berubah. Kondisi ini hanya berlaku apabila seluruh bobot sisi bernilai non-negatif.

Analisis Kompleksitas Waktu dan Ruang

Kompleksitas algoritma Dijkstra dipengaruhi oleh struktur data yang digunakan untuk menyimpan graph dan memilih simpul dengan jarak minimum

Kompleksitas Waktu

1. Menggunakan adjacency matrix:
   $\boldsymbol{T(V) = O(V^2)}$
   Dengan:
   - **T(V)**: waktu eksekusi algoritma sebagai fungsi dari jumlah simpul pada graph
   - **V (Vertex)**: jumlah simpul pada graph
   - **O (Big-O)**: notasi yang menyatakan batas atas laju pertumbuhan waktu eksekusi algoritma terhadap ukuran input
2. Menggunakan adjacency list dan *binary heap*:
   T(V,E)=O((V + E) log V)
   Dengan:

- **T(V,E)**: waktu eksekusi algoritma sebagai fungsi dari jumlah simpul dan jumlah sisi pada graph
- **E (Edge)**: jumlah sisi pada graph
- **log V**: kompleksitas operasi pada struktur data *heap* dalam proses pengambilan simpul dengan jarak minimum

3. Menggunakan *Fibonacci heap*:
   T(V,E)=O(E + V log V)
   Namun, implementasi *Fibonacci heap* jarang digunakan dalam praktik karena kompleksitas konstanta yang tinggi.
   Kompleksitas Ruang
   Kompleksitas ruang bergantung pada representasi graph:
   - Adjacency matrix: $O(V^2)$
   - Adjacency list: O(V + E)

   Untuk graph berskala besar, adjacency list lebih efisien karena hanya menyimpan sisi yang benar-benar ada.

Studi Kasus Perhitungan Algoritma Dijkstra

Untuk memperjelas proses kerja algoritma Dijkstra, digunakan sebuah graph berbobot sederhana yang terdiri dari tiga simpul. Studi kasus ini bertujuan untuk menunjukkan tahapan pemilihan simpul dengan jarak minimum dan proses relaksasi hingga diperoleh jalur terpendek dari simpul sumber ke simpul tujuan.

Contoh:

Misalkan diberikan graph berbobot dengan himpunan simpul:

V={A,B,C}

dan himpunan sisi:

V={(A.B),(A,C),(B,C)}

dengan bobot masing-masing sisi sebagai berikut:

SISI	BOBOT
A	4
B	2
C	1

Simpul A ditentukan sebagai simpul sumber Tujuan perhitungan adalah menentukan jalur terpendek dari simpul A ke simpul lainnya yaitu B dan C

Pada tahap awal, jarak dari simpul sumber ke dirinya sendiri bernilai nol, sedangkan jarak ke simpul lain bernilai tak hingga (∞):

- d(A)=0
- d(B)=∞
- d(C)=∞

Simpul dengan jarak minimum dipilih, yaitu A dengan nilai:

d(A)=0

Dilakukan relaksasi terhadap simpul yang bertetangga dengan A:

d(B)=min(∞,0+4)=4

d(C)=min(∞,0+2)=2

Hasil sementara:

SISI	BOBOT
A	0
B	4
C	2

Simpul dengan jarak minimum berikutnya adalah C dengan nilai:
d(C)=2
Dilakukan relaksasi terhadap simpul B melalui C:
d(B)=min(4,2+1)=3
Hasil sementara:

SISI	BOBOT
A	0
B	3
C	2

Simpul terakhir yang dipilih adalah B dengan nilai:
d(B)=3

TUJUAN	JARAK	JALUR
A	0	A
B	3	A → C → B
C	2	A → C

Studi kasus ini menunjukkan bahwa algoritma Dijkstra mampu menentukan jalur terpendek secara optimal dengan jumlah iterasi yang sebanding dengan jumlah simpul pada graph.

Analisis Keterbatasan terhadap Bobot Negatif

Pada graph dengan bobot negatif, asumsi dasar algoritma Dijkstra tidak terpenuhi. Misalkan terdapat graph:

$$A \rightarrow B = 4$$
$$A \rightarrow C = 2$$
$$C \rightarrow B = -5$$

Secara optimal:

$$A \rightarrow C \rightarrow B = -3$$

Namun, algoritma Dijkstra menetapkan B terlebih dahulu dengan jarak 4 sebelum mempertimbangkan jalur melalui C, sehingga menghasilkan solusi yang tidak optimal. Hal ini menunjukkan bahwa algoritma Dijkstra memerlukan *monotonicity property*, yaitu nilai jarak minimum tidak boleh berkurang setelah suatu simpul dipilih.

Ketidakmampuan Mendeteksi Negative Cycle

Negative cycle merupakan suatu siklus pada graph yang memiliki total bobot negatif. Keberadaan siklus ini menyebabkan nilai jarak terpendek menjadi tidak terdefinisi karena jarak dapat terus diperkecil dengan mengelilingi siklus tersebut secara berulang. Algoritma Dijkstra tidak memiliki mekanisme untuk mendeteksi adanya negative cycle karena proses relaksasi hanya dilakukan satu kali untuk setiap simpul. Hal ini berbeda dengan algoritma Bellman-Ford yang melakukan relaksasi sebanyak $|V| - 1$ kali sehingga mampu mengidentifikasi adanya perubahan nilai jarak setelah iterasi maksimum. Keterbatasan ini menunjukkan bahwa algoritma Dijkstra hanya sesuai digunakan pada graph dengan bobot non-negatif dan tidak mengandung siklus negatif.

Keterbatasan pada Graph Skala Besar

Secara teoritis, kompleksitas waktu algoritma Dijkstra adalah:

- $O(V^2)$ untuk implementasi matriks ketetanggaan
- $O((V + E) \log V)$ menggunakan priority queue

Meskipun kompleksitas ini relatif efisien untuk graph dengan ukuran sedang, performa algoritma dapat menurun secara signifikan pada graph berskala sangat besar (large-scale graph), seperti pada jaringan sosial atau sistem navigasi global. Permasalahan utama yang muncul meliputi:

1. Konsumsi memori yang tinggi pada representasi matriks.
2. Waktu eksekusi yang meningkat seiring pertambahan jumlah simpul dan sisi.
3. Ketergantungan pada struktur data heap yang efisien.

Analisis Sifat Greedy pada Algoritma Dijkstra

Algoritma Dijkstra termasuk ke dalam kategori algoritma greedy karena pada setiap langkahnya memilih solusi lokal terbaik dengan harapan menghasilkan solusi global yang optimal. Dalam konteks pencarian jalur terpendek pada graph, strategi greedy yang digunakan adalah memilih simpul dengan jarak minimum sementara dari simpul sumber dan menetapkannya sebagai jarak final

Konsep Greedy Choice Property

Sifat utama dari algoritma greedy adalah *greedy choice property*, yaitu setiap keputusan yang diambil pada suatu langkah merupakan pilihan terbaik berdasarkan kondisi saat itu tanpa mempertimbangkan kembali keputusan yang telah diambil sebelumnya. Pada algoritma Dijkstra, greedy choice dilakukan pada saat pemilihan simpul dengan nilai jarak minimum. Simpul tersebut dianggap telah memiliki jarak terpendek yang bersifat final sehingga tidak akan diperbarui kembali pada iterasi berikutnya. Secara matematis, jika terdapat himpunan simpul yang belum diproses, maka simpul u yang dipilih memenuhi kondisi: $d(u)=\min\{d(v)\}$

Monotonicity Property sebagai Syarat Keoptimalan

Keberhasilan pendekatan greedy pada algoritma Dijkstra bergantung pada terpenuhinya *monotonicity property*, yaitu nilai jarak minimum tidak akan berkurang setelah suatu simpul dipilih. Sifat ini hanya berlaku apabila seluruh bobot sisi bernilai non-negatif. Ketika bobot sisi bernilai negatif, jalur yang awalnya lebih panjang dapat menjadi lebih pendek setelah melalui sisi tersebut. Kondisi ini menyebabkan keputusan greedy yang telah diambil sebelumnya menjadi tidak valid. Dengan demikian, dapat disimpulkan bahwa:

- Jika bobot sisi $\geq 0 \rightarrow$ *greedy* valid
- Jika terdapat bobot negatif $\rightarrow$ *greedy* tidak valid

Keunggulan Pendekatan Greedy pada Dijkstra

Pendekatan greedy memberikan beberapa keunggulan, antara lain:

1. Proses pemilihan simpul dilakukan secara sederhana dan sistematis.
2. Tidak memerlukan pemeriksaan ulang terhadap simpul yang telah diproses.
3. Menghasilkan kompleksitas waktu yang lebih efisien dibandingkan pendekatan brute force.
4. Cocok digunakan pada graph berskala besar dengan bobot non-negatif.

Hal ini menyebabkan algoritma Dijkstra banyak digunakan dalam berbagai aplikasi seperti sistem navigasi, jaringan komputer, dan sistem logistik.

Keterbatasan Pendekatan Greedy

Meskipun efisien, pendekatan greedy memiliki keterbatasan karena keputusan yang telah diambil tidak dapat direvisi kembali. Pada algoritma Dijkstra, simpul yang telah ditetapkan sebagai final tidak akan diperbarui meskipun ditemukan jalur yang lebih pendek pada iterasi selanjutnya. Kondisi ini menjadi permasalahan utama ketika:

1. Graph memiliki bobot sisi negatif.
2. Graph bersifat dinamis (bobot berubah).
3. Diperlukan evaluasi ulang terhadap jalur yang telah dipilih

Analisis Teoretis terhadap Validitas Greedy

Secara teoretis, algoritma Dijkstra dapat menghasilkan solusi optimal apabila memenuhi kondisi berikut:

1. Graph tidak mengandung bobot negatif.
2. Memenuhi optimal substructure.
3. Memenuhi greedy choice property.

Jika salah satu kondisi tersebut tidak terpenuhi, maka pendekatan greedy tidak dapat menjamin keoptimalan solusi.

Implementasi Algoritma Dijkstra

Untuk mendukung hasil analisis teoritis yang telah dilakukan, penelitian ini dilengkapi dengan implementasi algoritma Dijkstra menggunakan bahasa pemrograman Python. Implementasi ini bertujuan untuk mensimulasikan proses pencarian jalur terpendek secara komputasional serta menguji keakuratan hasil yang diperoleh dari perhitungan manual. Algoritma diimplementasikan menggunakan struktur data priority queue (heap) untuk meningkatkan efisiensi dalam proses pemilihan simpul dengan jarak minimum. Representasi graph menggunakan adjacency list dalam bentuk dictionary, di mana setiap simpul memiliki pasangan simpul tetangga beserta bobot sisinya. Berikut merupakan implementasi algoritma Dijkstra yang digunakan dalam penelitian ini:

```
import heapq

def dijkstra(graph, start):
 distances = {node: float('inf') for node in graph}
 distances[start] = 0

 previous = {node: None for node in graph}

 pq = [(0, start)]

 while pq:
 current_distance, current_node = heapq.heappop(pq)

 for neighbor, weight in graph[current_node].items():
 distance = current_distance + weight

 if distance < distances[neighbor]:
 distances[neighbor] = distance
 previous[neighbor] = current_node
 heapq.heappush(pq, (distance, neighbor))

 return distances, previous

def get_path(previous, target):
 path = []
 while target is not None:
 path.insert(0, target)
 target = previous[target]
 return path
```

```
graph = {
 'A': {'B': 4, 'C': 2},
 'B': {},
 'C': {'B': 1}
}

distances, previous = dijkstra(graph, 'A')

print("="*40)
print("HASIL DIJKSTRA DARI NODE A")
print("="*40)

for node in graph:
 path = get_path(previous, node)
 print(f"\nTujuan Node : {node}")
 print(f"Jarak Minimum : {distances[node]}")
 print(f"Jalur Terpendek : {' -> '.join(path)}")
 print("-"*40)

 print("Jarak terpendek dari A:")
for node in sorted(graph):
 path = get_path(previous, node)
 print(f"A -> {node} = {distances[node]} | Jalur: {' -> '.join(path)}")

print("\n")
```

Penjelasan kode program:

Kode program di atas merupakan implementasi algoritma Dijkstra untuk menentukan jarak terpendek dari satu simpul sumber ke simpul lainnya pada suatu graf berbobot. Algoritma ini memanfaatkan struktur data *priority queue* yang diimplementasikan menggunakan library heapq untuk meningkatkan efisiensi proses pencarian. Fungsi dijkstra(graph, start) digunakan untuk menghitung jarak minimum dari simpul awal (start) ke seluruh simpul dalam graf. Pada awal fungsi, dilakukan inisialisasi nilai jarak semua simpul dengan nilai tak hingga (infinity), kemudian jarak simpul awal diatur menjadi 0. Selain itu, dibuat dictionary previous untuk menyimpan simpul sebelumnya pada jalur terpendek.

```
...  ========================================
     HASIL DIJKSTRA DARI NODE A
     ========================================

     Tujuan Node : A
     Jarak Minimum : 0
     Jalur Terpendek : A
     ----------------------------------------
     Jarak terpendek dari A:

     Tujuan Node : B
     Jarak Minimum : 3
     Jalur Terpendek : A -> C -> B
     ----------------------------------------
     Jarak terpendek dari A:

     Tujuan Node : C
     Jarak Minimum : 2
     Jalur Terpendek : A -> C
     ----------------------------------------
     Jarak terpendek dari A:
     A -> A = 0 | Jalur: A
     A -> B = 3 | Jalur: A -> C -> B
     A -> C = 2 | Jalur: A -> C
```

Priority queue (pq) digunakan untuk menyimpan pasangan nilai jarak dan simpul yang akan diproses. Selama queue tidak kosong, algoritma akan mengambil simpul dengan jarak terkecil, kemudian mengevaluasi setiap tetangganya. Jika ditemukan jarak yang lebih pendek

melalui simpul tersebut, maka nilai jarak akan diperbarui dan simpul sebelumnya dicatat. Simpul tetangga tersebut kemudian dimasukkan kembali ke dalam priority queue untuk diproses lebih lanjut. Fungsi get_path(previous, target) digunakan untuk merekonstruksi jalur terpendek dari simpul awal ke simpul tujuan (target). Proses ini dilakukan dengan menelusuri kembali simpul sebelumnya hingga mencapai simpul awal, kemudian menyusunnya menjadi urutan jalur yang benar. Graf direpresentasikan dalam bentuk dictionary, di mana setiap simpul memiliki pasangan simpul tetangga beserta bobotnya. Pada contoh ini, simpul A terhubung ke B dan C, sedangkan simpul C terhubung ke B. Setelah fungsi dijalankan, program mencetak hasil berupa jarak minimum dan jalur terpendek dari simpul A ke setiap simpul tujuan. Output ditampilkan dalam dua format, yaitu per simpul secara rinci dan dalam bentuk ringkasan keseluruhan. Berdasarkan implementasi tersebut, dapat disimpulkan bahwa program telah berhasil menerapkan algoritma Dijkstra secara efektif untuk menentukan jalur terpendek pada graf berbobot.

Hasil Eksekusi Program

Setelah dilakukan eksekusi program, diperoleh hasil sebagai berikut: Hasil keluaran algoritma Dijkstra menunjukkan bahwa simpul awal yang digunakan adalah simpul A dengan jarak awal bernilai 0. Berdasarkan proses perhitungan, diperoleh bahwa jarak terpendek dari simpul A ke simpul B adalah sebesar 3 dengan rute A → C → B, yang menandakan bahwa jalur tidak langsung memberikan jarak yang lebih optimal dibandingkan jalur lainnya. Selain itu, jarak terpendek dari simpul A ke simpul C adalah sebesar 2 melalui jalur langsung A → C. Secara keseluruhan, hasil ini memperlihatkan bahwa algoritma berhasil menentukan jalur terpendek dari simpul sumber ke setiap simpul tujuan dengan tepat. Nilai jarak dan rute yang dihasilkan juga konsisten dengan perhitungan manual yang telah dilakukan sebelumnya, sehingga dapat disimpulkan bahwa implementasi algoritma Dijkstra pada sistem telah berjalan dengan benar dan sesuai dengan teori.

Analisis Hasil Implementasi

Berdasarkan hasil implementasi, dapat dianalisis bahwa algoritma Dijkstra mampu menghasilkan solusi yang konsisten dengan perhitungan manual pada graph berbobot non-negatif. Hal ini menunjukkan bahwa asumsi dasar algoritma, yaitu terpenuhinya greedy choice property dan monotonicity property, berlaku pada kasus yang diuji. Penggunaan priority queue dalam implementasi juga menunjukkan efisiensi algoritma dalam memilih simpul dengan jarak minimum, sehingga kompleksitas waktu dapat ditekan menjadi lebih optimal dibandingkan dengan pendekatan sederhana. Selain itu, implementasi ini memperlihatkan bahwa proses relaksasi berjalan secara sistematis dan menghasilkan pembaruan jarak yang konsisten hingga mencapai solusi optimal. Dengan demikian, hasil implementasi memperkuat analisis teoritis bahwa algoritma Dijkstra efektif digunakan pada graph berbobot non-negatif, namun tetap memiliki keterbatasan sebagaimana telah dijelaskan pada subbab sebelumnya.

KESIMPULAN

Berdasarkan hasil analisis teoritis yang dilakukan, dapat disimpulkan bahwa algoritma Dijkstra merupakan metode yang efektif dan optimal untuk menyelesaikan permasalahan single source shortest path pada graph berbobot non-negatif. Keoptimalan algoritma bergantung pada terpenuhinya sifat greedy choice property, optimal substructure, dan monotonicity property. Namun, penelitian ini menegaskan bahwa algoritma Dijkstra memiliki keterbatasan mendasar. Algoritma tidak dapat menangani graph dengan bobot sisi negatif dan tidak mampu mendeteksi negative cycle, sehingga tidak selalu menghasilkan solusi optimal pada kondisi tertentu. Selain itu, pada graph berskala besar maupun graph dinamis, efisiensi

algoritma dapat menurun karena meningkatnya kompleksitas komputasi dan kebutuhan perhitungan ulang. Dengan demikian, tidak ada algoritma shortest path yang bersifat universal untuk semua kondisi graph. Pemilihan algoritma harus mempertimbangkan karakteristik struktur graph, jenis bobot, serta kebutuhan komputasi. Penelitian ini memberikan kontribusi berupa pemahaman komprehensif mengenai batasan operasional algoritma Dijkstra sebagai dasar pertimbangan dalam pemilihan algoritma yang tepat pada berbagai aplikasi komputasi. Selain itu, implementasi algoritma menggunakan Python menunjukkan bahwa hasil komputasi sejalan dengan analisis teoritis, sehingga memperkuat validitas algoritma Dijkstra dalam menyelesaikan permasalahan shortest path pada graph berbobot non-negatif.

DAFTAR PUSTAKA

Aldhafferi, N. (2025). Time and memory trade-offs in shortest-path algorithms across graph topologies: A*, Bellman–Ford, Dijkstra, AI-augmented A* and a neural baseline. *Computers, 14*, 545.

Amin, A., & Hendrik, B. (2023). Analisis penerapan algoritma Dijkstra dalam optimasi penentuan rute: Sebuah kajian literatur sistematis. *Journal of Education Research, 6*(1).

Angul, A., Fallo, D., Tanggo, K. V., Belo, I. N. A., & Hoar, F. (2025). Implementasi algoritma Dijkstra dan Greedy dalam penyelesaian masalah rute terpendek. *Jurnal Kridatama Sains dan Teknologi, 7*(01), 489–496.

Asmiati, P., Prawinasti, K., Damayanti, M., & Yulianti, L. (2025). The locating chromatic number of (k, n)-split cycle graph and its barbell operation. *Electronic Journal of Graph Theory and Applications, 13*(2), 249–258.

Bhaskara, I. M. A., Kumara, I. M. S., Darma, I. G. W., & Raharja, I. K. A. W. (2024). Perbandingan algoritma Dijkstra dan Floyd-Warshall menggunakan software defined network untuk rute terpendek. *RESISTOR, 7*(1), 109–118.

Chen, R. (2022). Dijkstra's shortest path algorithm and its application on bus routing. In *Proceedings of the 2022 International Conference on Urban Planning and Regional Economy (UPRE 2022)*.

Duan, R., Mao, J., Mao, X., Shu, X., & Yin, L. (2025). Breaking the sorting barrier for directed single-source shortest paths. *Manuscript*.

Durand, A., Watteau, T., Ghazi, G., & Botez, R. M. (2024). Generalized shortest path problem: An innovative approach for non-additive problems in conditional weighted graphs. *Mathematics, 12*, 2995.

Fedorov, D., Kontsevik, G., Bashirov, R., Mityagin, S., Tupikina, L., Zakharenko, N., & Budennyy, S. (2025). Assessing the complexity of a path search optimization method based on clustering for a transport graph. *EPJ Data Science, 14*, 32.

Hadi, H. M., & Ibrahim, I. M. (2025). A comprehensive review of shortest path algorithms for network routing. *Asian Journal of Research in Computer Science, 18*(3), 152–175.

Haritha, T., & Chithra, A. V. (2025). The distance Seidel matrix of connected graphs. *AKCE International Journal of Graphs and Combinatorics, 22*(3), 241–252.

Hoepner, J. I., MacGillivray, G., & Mynhardt, C. M. (2025). Maximum boundary independent broadcasts in graphs and trees. *Electronic Journal of Graph Theory and Applications, 13*(2), 237–248.

Iranmanesh, M. A., & Moghaddami, N. (2025). Some properties of Cayley signed graphs on finite Abelian groups. *Electronic Journal of Graph Theory and Applications, 13*(2), 417–422.

Jelita, F., Fallo, D., & Miru, Y. G. (2025). Optimalisasi rute menggunakan algoritma Dijkstra dan Greedy: Sebuah pendekatan komparatif. *Jurnal Kridatama Sains dan Teknologi, 7*(01), 555–562.

Kalita, J., & Paul, S. (2025). On adjacency and (signless) Laplacian spectra of centralizer and co-centralizer graphs of some finite non-abelian groups. *Electronic Journal of Graph Theory and Applications, 13*(2), 397–416.

Kamyab, K., Ghasemi, M., & Varmazyarb, R. (2025). Minimally 4-restricted edge connected graphs. *Electronic Journal of Graph Theory and Applications, 13*(2), 375–382.

Kim, K. (2025). The Italian bondage and reinforcement numbers of digraphs. *Electronic Journal of Graph Theory and Applications, 13*(2), 361–374.

Korivand, M., Soltankhah, N., & Khashyarmanesh, K. (2025). Uniquely proper distinguishing colorable graphs. *Electronic Journal of Graph Theory and Applications, 13*(2), 423–440.

Lewis, R. (2020). Algorithms for finding shortest paths in networks with vertex transfer penalties. *Algorithms, 13*, 269.

Liu, Y., Lin, Q., Hong, B., Peng, Y., Hjerpe, D., & Liu, X. (2023). Resonance algorithm: An intuitive algorithm to find all shortest paths between two nodes. *Complex & Intelligent Systems, 9*, 4159–4167.

Madkour, A., Aref, W. G., Rehman, F. U., Rahman, M. A., & Basalamah, S. (2017). A survey of shortest-path algorithms. *arXiv preprint arXiv:1705.02044*.

Mane, S. A., & Shinde, N. V. (2025). Quasi perfect codes in the cartesian product of some graphs. *Electronic Journal of Graph Theory and Applications, 13*(2), 441–458.

Mojdeh, D. A., & Samadzadeh, M. R. (2025). Perfect coalition in graphs. *Electronic Journal of Graph Theory and Applications, 13*(2), 345–360.

More, R. D., Patil, A. S., Dandwate, D. S., & Tupe, U. J. (2021). A literature review on applications of graph theory in various fields. *IJRTI, 6*(1), 1–8.

Musridho, R. J., Rusnedy, H., & Sudirman, W. F. R. (2025). Analisis komparatif algoritma Dijkstra dan Google Maps API dalam penentuan rute tercepat. *Journal of Artificial Intelligence and Digital Business (RIGGS), 4*(3), 5017–5022.

Pancahayani, S., Simanjuntak, R., Baca, M., Semanicova-Fenovcíkova, A., & Uttunggadewa, S. (2025). On the relations among edge magic total, edge antimagic total, and ASD-antimagic graphs. *Electronic Journal of Graph Theory and Applications, 13*(2), 387–396.

Rachmawati, D., & Gustin, L. (2020). Analysis of Dijkstra's algorithm and A* algorithm in shortest path problem. *Journal of Physics: Conference Series, 1566*(1).

Riti, Y. F., Iskandar, J. S., & Hendra. (2023). Comparison analysis of graph theory algorithms for shortest path problem.

Salsabila, T. N., Chaerani, D., & Napitupulu, H. (2026). Systematic literature review of GPS-based multi-objective environmentally friendly shortest path with a proposed lexicographic framework. *CAUCHY – Jurnal Matematika Murni dan Aplikasi, 11*(1), 364–381.

Sinlae, D. A., Ledoh, P. K., & Fallo, D. (2025). Analisis algoritma Dijkstra dan Greedy pada pencarian jalur terpendek di graf berbobot. *Jurnal Penelitian Multidisiplin Terpadu, 9*(6), 282–286.

Young, N. E., Tarjan, R. E., & Orlin, J. B. (1991). Faster parametric shortest path and minimum balance algorithms. *Networks, 21*(2).