**Jurnal Sifo Mikroskil (JSM)**
**Volume 27, No 1, April 2026 – Hal 127 - 146**
**DOI: https://doi.org/10.55601/jsm.v27i1.1941**

**e-ISSN: 2622-8130**
**ISSN: 1412-0100**

# Analisis Komparatif Pola Traversal BFS dan DFS pada Graf Jaringan Jalan Nyata untuk Sistem Distribusi Kurir

**Topan Bagus Prasetyo [1], Ni Luh Dewi Sintiari [2], Kristian Telaumbanua [3]**
[1,2]Universitas Pendidikan Ganesha, Jl. Udayana No.11, Telp (0362) 22571
[3]Universitas Mikroskil, Jl. Thamrin No. 112, 124, 140, Telp (061) 4567789
[1,2]Teknik dan Kejuruan, Ilmu Komputer, Universitas Pendidikan Ganesha, Bali
[3]Informatika, Teknik Informatika, Universitas Mikroskil, Medan
e-mail: [1]topan@student.undiksha.ac.id., [2]luh.dewi.sintiari@undiksha.ac.id, [3]kristian@mikroskil.ac.id

*Dikirim:* ***dd-mm-yyyy*** | *Diterima:* ***dd-mm-yyyy*** | *Diterbitkan:* ***dd-mm-yyyy***

**Abstrak**

Penentuan rute optimal sangat esensial untuk efisiensi pengiriman logistik, terutama di wilayah dengan struktur jaringan jalan yang kompleks dan tidak teratur seperti Kaliuntu, Bali. Secara teoretis, algoritma *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS) memiliki kompleksitas waktu identik yakni $O(V + E)$. Namun, kajian performa empiris beserta analisis pola penelusurannya pada graf jaringan nyata masih sangat terbatas. Penelitian ini bertujuan membandingkan pola *traversal* spasial dan waktu eksekusi empiris kedua algoritma. Graf tak berarah dan tak berbobot (15 simpul, 20 sisi) hasil pemetaan lapangan diimplementasikan menggunakan Python 3.10 berbasis *adjacency list* (Intel Core i5-7200U, RAM 8GB). Eksperimen melalui 10 iterasi menunjukkan komputasi BFS lebih cepat dan stabil (rata-rata 0.082 ms, 6 rute unik) dibandingkan DFS (rata-rata 0.158 ms, 9 rute unik). Namun, analisis spasial mengungkap sebuah paradoks operasional yaitu pola radial BFS memaksa kurir melompat antar klaster secara tidak efisien, sedangkan strategi linier DFS jauh lebih rasional karena menuntaskan satu kawasan utuh sebelum berpindah. Temuan ini menegaskan bahwa pemilihan algoritma logistik wajib menyelaraskan logika algoritmik dengan intuisi spasial kurir manusia di lapangan.

**Kata kunci**: Breadth_First_Search, Depth_First_Search, Pola_Traversal, Graf, Rute_Kurir

***Abstract***

*Determining optimal routes is highly essential for logistics delivery efficiency, especially in areas with complex and irregular road networks like Kaliuntu, Bali. Theoretically, Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms share an identical time complexity of $O(V + E)$. However, empirical performance studies and traversal pattern analyses on real-world network graphs remain very limited. This study aims to compare the spatial traversal patterns and empirical execution times of both algorithms. An undirected and unweighted graph (15 vertices, 20 edges) derived from field mapping was implemented using Python 3.10 with an adjacency list structure (Intel Core i5-7200U, 8GB RAM). Experimental results across 10 iterations showed BFS is computationally faster and more stable (0.082 ms average, 6 unique routes) than DFS (0.158 ms average, 9 unique routes). However, spatial analysis revealed a significant operational paradox that is BFS's radial pattern forces inefficient jumping between geographical clusters, whereas DFS's linear strategy is much more rational as it completes one entire area before moving. These findings confirm that selecting logistics algorithms must seamlessly align algorithmic logic with the spatial intuition of human couriers in the field.*

***Keywords****: Breadth_First_Search, Depth_First_Search, Traversal_Pattern, Graph, Courier_Route*

## 1. PENDAHULUAN

Pertumbuhan *e-commerce* di Indonesia meningkat pesat dalam satu dekade terakhir, seiring dengan perubahan perilaku masyarakat yang beralih ke belanja daring karena faktor kemudahan, kecepatan, dan efisiensi biaya [1] Perubahan ini berdampak langsung pada meningkatnya kebutuhan akan layanan logistik dan kurir ekspedisi yang mampu mengantarkan paket secara cepat dan tepat waktu [2]. Dengan begitu, efisiensi penentuan rute pengiriman menjadi salah satu faktor penentu kualitas layanan. Permasalahan signifikan dalam layanan kurir di perkotaan Indonesia muncul akibat kompleksitas jaringan jalan yang ditandai dengan pola jalan tidak teratur, gang sempit yang saling terhubung, serta hambatan lalu lintas pada jalur utama pada jam sibuk. Kondisi ini sering menyebabkan kurir menempuh rute yang kurang efisien, sehingga waktu pengantaran lebih lama dan biaya operasional meningkat. Fenomena ini sejalan dengan penelitian yang menekankan bahwa perencanaan rute transportasi dan pengiriman yang optimal merupakan faktor kritis dalam meningkatkan efisiensi biaya logistik sekaligus memperpendek waktu transit dalam sistem transportasi di Indonesia [3].

Permasalahan rute pengiriman dapat dimodelkan secara matematis menggunakan graf, yang secara formal didefinisikan sebagai pasangan himpunan $G = (V(G), E(G))$, dengan $V(G)$ merupakan himpunan simpul (*vertex*) tak kosong yang merepresentasikan titik-titik penting seperti jalan utama atau persimpangan, sedangkan $E(G)$ merupakan himpunan sisi (*edge*) yang melambangkan jalur penghubung antar lokasi [4]. Dalam konteks aplikasi graf, objek direpresentasikan dengan simpul, sedangkan hubungan antar objek direpresentasikan dengan sisi, sehingga struktur graf mampu merepresentasikan jaringan jalan yang kompleks. Dengan pendekatan ini, pencarian rute optimal dapat dilakukan menggunakan algoritma penelusuran graf.

Algoritma *Breadth-First Search* (BFS) dikenal efektif dalam menemukan jalur terpendek pada graf tak berbobot karena sifat penjelajahannya yang sistematis melebar dari simpul awal, sementara *Depth-First Search* (DFS) lebih menekankan pencarian secara mendalam. Secara matematis dan teoretis, kedua algoritma dasar ini memiliki kompleksitas waktu yang sama persis, yaitu $O(V + E)$. Namun, tinjauan literatur menunjukkan bahwa sebagian besar analisis performa BFS dan DFS umumnya dievaluasi pada struktur graf buatan (*synthetic graphs*) atau lingkungan simulasi terkontrol, seperti grid labirin (*maze*) [5]. Masih sangat sedikit literatur yang secara spesifik mengkaji performa empiris dan perilaku penelusuran kedua algoritma ini ketika dihadapkan pada karakteristik jaringan jalan nyata (*real-world network*) yang kompleks, asimetris, dan tidak teratur.

Berdasarkan hal tersebut, penelitian ini memfokuskan pada studi komparatif pola *traversal* algoritma BFS dan DFS pada graf jaringan jalan nyata. Pengukuran waktu eksekusi (*running time*) pada artikel ini bukanlah indikator empiris mutlak terhadap performa kedua algoritma. Sebaliknya, waktu eksekusi dikaji murni sebagai instrumen pembanding terhadap kompleksitas teoretis $O(V + E)$ dengan menggunakan model komputasi yang sama. Selain membandingkan waktu komputasi, studi ini dititikberatkan pada kajian pola *traversal* yang dibentuk oleh kedua algoritma dan kesesuaiannya dengan ritme serta strategi kerja operasional kurir di lapangan.

Wilayah Kaliuntu di Kabupaten Buleleng, Bali dipilih sebagai lokasi studi kasus karena memiliki karakteristik jaringan jalan yang kompleks dengan banyak gang sempit, jalan satu arah, dan persimpangan tidak beraturan. Kompleksitas ini mencerminkan tantangan nyata yang dihadapi kurir ekspedisi di wilayah perkotaan Indonesia. Adapun rumusan masalah penelitian ini adalah: bagaimana komparasi pola *traversal* dan efisiensi algoritma BFS dan DFS pada graf jaringan jalan nyata di wilayah Kaliuntu untuk penentuan rute kurir ekspedisi? Tujuan penelitian ini yaitu untuk menganalisis dan membandingkan kinerja empiris serta perilaku penelusuran BFS dan DFS, sehingga dapat dievaluasi kesesuaian algoritma tersebut dengan strategi operasional kurir di daerah berkarakteristik jalan yang kompleks.

Untuk menjaga fokus penelitian, beberapa batasan ditetapkan. Representasi graf dibangun sebagai graf tak berarah dan tak berbobot, dengan simpul hanya merepresentasikan titik-titik penting hasil observasi dan pemetaan wilayah. Pemilihan fokus ini selaras dengan pemanfaatan data spasial dalam bidang analisis dan pemodelan, sehingga graf digunakan sebagai teknik representasi jaringan jalan [6].

Faktor eksternal seperti lalu lintas, kondisi cuaca, maupun kebijakan jalan tidak dipertimbangkan, karena penelitian ini menitikberatkan pada karakteristik algoritmik murni. Selain itu, analisis dibatasi pada kompleksitas waktu tanpa membahas kompleksitas ruang, serta implementasi dilakukan menggunakan bahasa pemrograman Python tanpa integrasi ke sistem navigasi *real-time*.

## 2. TINJAUAN PUSTAKA

### 2.1 Kompleksitas Waktu

Kompleksitas waktu merupakan ukuran utama efisiensi algoritma yang menunjukkan jumlah operasi yang dibutuhkan seiring bertambahnya ukuran input, dan umumnya direpresentasikan dengan Notasi $Big\ O$ [7]. Karena waktu eksekusi yang efisien secara langsung memengaruhi kecepatan pengiriman paket, kompleksitas waktu menjadi indikator utama untuk menilai kinerja algoritma kurir ketika berfokus pada sistem logistik dan penentuan rute [8]. Secara teoretis, algoritma penjelajahan graf seperti *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS) memiliki kompleksitas waktu yang sama, yaitu $O(V + E)$, $V$ adalah jumlah simpul (*vertex*) dan $E$ adalah jumlah sisi (*edge*) graf [9].

Penelitian terdahulu menyoroti bahwa perbandingan kompleksitas algoritma tidak hanya penting untuk menilai waktu eksekusi, tetapi juga untuk memahami pengaruh struktur data terhadap performa *traversal*. Misalnya, perbandingan algoritma Kruskal, Prim, dan Sollin dalam menentukan *Minimum Spanning Tree* menunjukkan bahwa efisiensi algoritma sangat bergantung pada cara data direpresentasikan [10], [11]. Dengan analogi serupa, analisis terhadap BFS dan DFS menjadi penting karena keduanya menggunakan strategi *traversal* berbeda yang dapat menghasilkan performa bervariasi pada jenis graf tertentu. Dengan demikian, pemahaman kompleksitas waktu menjadi dasar teoretis dalam menilai efisiensi kedua algoritma tersebut sebelum diterapkan pada model graf jaringan jalan di wilayah penelitian. Analisis perbandingan empiris terhadap kompleksitas waktu algoritma BFS dan DFS menjadi langkah penting untuk memvalidasi efektivitas keduanya dalam skenario operasional nyata. Efisiensi komputasi dari kedua algoritma tersebut sangat dipengaruhi oleh struktur serta representasi graf yang digunakan, sehingga pemahaman mendalam mengenai teori graf menjadi dasar penting dalam menilai kinerja *traversal* pada sistem penentuan rute kurir di Kaliuntu.

### 2.2 Graf

Graf merupakan struktur fundamental dalam ilmu komputer yang digunakan untuk merepresentasikan hubungan antar entitas melalui simpul (*vertex*) dan sisi (*edge*). Jika sisi-sisi pada graf memiliki bobot berupa bilangan riil, maka graf tersebut disebut sebagai graf berbobot, dan jika sisi-sisi graf memiliki orientasi tertentu, maka graf tersebut disebut graf berarah. Melalui struktur ini, hubungan kompleks antar elemen dalam suatu sistem dapat dimodelkan secara sistematis dan dianalisis secara matematis. Teori graf telah terbukti menjadi landasan penting untuk pemecahan masalah nyata dalam berbagai bidang, mulai dari perancangan jaringan komunikasi, sistem transportasi, hingga pemetaan hubungan sosial, data digital berskala besar serta sistem logistik yang bergantung pada keterhubungan antar titik lokasi [12].

Dalam penentuan rute kurir, representasi graf memungkinkan setiap titik lokasi seperti gudang, jalan utama, atau tujuan pengiriman digambarkan sebagai simpul, dan jalur yang menghubungkannya digambarkan sebagai sisi. Representasi ini menjadikan graf sebagai model matematis yang mampu menggambarkan jaringan jalan secara sistematis serta mendukung analisis lintasan antar lokasi dengan efisien. Pemanfaatan emanfaatan graf dalam sistem informasi modern juga telah berkembang pesat untuk menganalisis pola keterhubungan yang mendukung optimasi proses *traversal* dan efisiensi pergerakan pada jaringan berskala besar [13]. Sehingga, efektivitas algoritma penjelajahan graf sangat bergantung pada cara graf direpresentasikan serta pada kompleksitas struktural dari graf itu sendiri [14], [15].

Dalam penelitian ini, teori graf digunakan untuk memodelkan jaringan jalan di Kaliuntu sebagai graf tak berarah dan tak berbobot guna menekankan analisis *traversal* murni. Representasi ini menjadi dasar dalam mengevaluasi kinerja algoritma *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS)

terhadap efisiensi waktu penelusuran. Dengan demikian, teori graf tidak hanya berperan sebagai landasan konseptual, tetapi juga sebagai kerangka operasional yang menghubungkan antara pemodelan jaringan dan analisis algoritmik yang akan dibahas pada subbab berikutnya.

### 2.3 Algoritma Penjelajahan Graf

Algoritma penjelajahan graf merupakan prosedur sistematis yang dirancang untuk mengunjungi setiap simpul dan sisi dalam struktur graf secara terurut. Algoritma ini menjadi fondasi utama dalam berbagai masalah komputasi yang melibatkan analisis keterhubungan, pencarian jalur, dan optimasi rute [13]. Dalam konteks penentuan rute kurir, masalah ini dapat dimodelkan sebagai pencarian jalur efisien pada graf yang merepresentasikan jaringan jalan di wilayah pengiriman. Secara lebih luas, masalah penentuan rute terpendek merupakan bagian integral dari persoalan optimasi kompleks seperti *Traveling Salesman Problem* (TSP), yaitu permasalahan bagi seorang kurir harus mengunjungi sejumlah tujuan dengan ketentuan setiap lokasi hanya dikunjungi sekali guna meminimalkan jarak, waktu, atau biaya perjalanan [16].

Untuk menyelesaikan masalah optimasi rute dan pencarian jalur, berbagai jenis algoritma telah dikembangkan dengan karakteristik dan strategi yang beragam. Metode yang umum digunakan dalam penyelesaian TSP antara lain algoritma *greedy, brute force, hill-climbing, ant colony optimization,* dan *genetic algorithm* [16]. Sementara itu, dalam lingkungan jaringan komunikasi, konsep serupa dikenal sebagai *routing protocol* yang berfungsi menentukan jalur pengiriman data antar perangkat. Protokol ini berperan penting dalam optimasi sistem jaringan, dan perbandingan performa antar protokol menjadi langkah krusial untuk meningkatkan efisiensi operasional [17]. Analogi konseptual ini menunjukkan bahwa prinsip penjelajahan graf tidak hanya relevan dalam konteks geografis, tetapi juga dalam berbagai domain yang melibatkan struktur keterhubungan.

Algoritma penjelajahan graf yang paling fundamental dan banyak digunakan adalah *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS). Kedua algoritma ini memanfaatkan struktur graf untuk mengunjungi semua simpul yang dapat dijangkau dari simpul awal dengan strategi penelusuran yang berbeda. BFS melakukan penjelajahan secara berlapis berdasarkan jarak dari simpul awal, sementara DFS melakukan penjelajahan secara mendalam mengikuti satu cabang hingga ujung sebelum berpindah ke cabang lain. Kompleksitas waktu dari kedua algoritma menjadi faktor kunci yang menentukan efisiensi penerapannya dalam aplikasi dunia nyata, termasuk sistem penentuan rute kurir. Perkembangan teori graf juga terus berlanjut dengan munculnya pendekatan yang lebih kompleks seperti *graph neural networks* dan *graph representation learning* untuk menganalisis serta mengintegrasikan hubungan data dalam skala besar [13]. Dengan demikian, pemahaman terhadap algoritma penjelajahan graf menjadi dasar penting sebelum menganalisis implementasi BFS dan DFS pada jaringan jalan di wilayah penelitian.

### 2.4 *Adjacency List*

Representasi struktur graf yang efisien merupakan faktor kunci dalam mencapai kompleksitas waktu optimal pada algoritma penjelajahan seperti BFS dan DFS. Dua simpul graf yang terhubung oleh sebuah sisi disebut bertetangga (*adjacent*). Terdapat beberapa cara merepresentasikan struktur graf, salah satunya adalah dengan daftar ketetanggaan (*adjacency list*). *Adjacency list* merupakan koleksi daftar yang menyimpan informasi simpul-simpul tetangga, dan untuk setiap simpul dalam graf $G = (V, E)$, sebuah daftar hanya menyimpan simpul lain yang terhubung langsung dengan simpul tersebut. Berbeda dengan matriks ketetanggaan (*adjacency matrix*) yang memerlukan ruang $O(|V|^2)$ tanpa mempertimbangkan jumlah sisi, *adjacency list* hanya membutuhkan ruang $O(|V| + |E|)$, sehingga sangat efisien untuk graf dengan jumlah sisi $|E|$ yang jauh lebih kecil dibandingkan kuadrat jumlah simpul ($|V|^2$) [14]. Oleh karena jaringan jalan kurir ekspedisi di Kaliuntu memiliki karakteristik sebagai graf jarang (*sparse graph*), maka *adjacency list* diadaptasi sebagai representasi yang paling optimal untuk struktur graf tersebut.

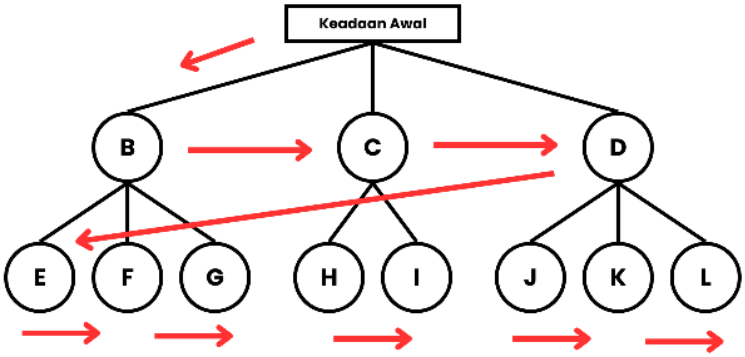
Pilihan representasi graf ini memiliki dampak langsung terhadap kinerja algoritma penjelajahan dan analisis kompleksitas waktu. Penelitian menunjukkan bahwa struktur representasi graf, seperti penggunaan matriks ketetanggaan, berpengaruh signifikan terhadap efisiensi proses *traversal* [14]. Ketika graf direpresentasikan dengan *adjacency list*, penjelajahan seluruh simpul dan sisi dapat dilakukan dengan kompleksitas waktu linear $O(|V| + |E|)$, karena setiap simpul dikunjungi tepat satu kali dan setiap sisi diperiksa hanya sekali selama proses penelusuran [18]. Efisiensi ini menjadi sangat penting dalam penelitian ini, mengingat pemodelan jaringan jalan yang besar memerlukan representasi yang mampu meminimalkan beban komputasi baik dari segi ruang maupun waktu, sehingga memungkinkan perbandingan performa algoritma yang akurat dan valid.

Dengan demikian, penggunaan *adjacency list* secara strategis mengoptimalkan memori (*space complexity*) dan, lebih jauh lagi, menjamin bahwa kinerja (*running time*) algoritma BFS dan DFS berbanding lurus dengan struktur simpul dan sisi yang ada di dalam graf. Keterkaitan linier ini merupakan landasan esensial guna memastikan validitas analisis komparatif kompleksitas waktu kedua algoritma dalam menyelesaikan masalah penentuan rute kurir ekspedisi.

### 2.5 *Breadth-First Search*

*Breadth-First Search* (BFS) merupakan algoritma penelusuran graf yang melakukan eksplorasi secara berlapis, dimulai dari simpul awal kemudian mengunjungi seluruh simpul tetangga sebelum melanjutkan ke level berikutnya. Algoritma ini memanfaatkan struktur data *queue* dengan prinsip *First In First Out* (FIFO), sehingga simpul yang pertama kali dimasukkan akan diproses terlebih dahulu [19]. Ciri khas utama BFS adalah kemampuannya menjamin bahwa jalur pertama yang ditemukan menuju simpul tujuan merupakan jalur terpendek dalam konteks graf tak berbobot. Hal ini terjadi karena algoritma selalu menjelajahi simpul dari tingkat terdekat menuju tingkat yang lebih jauh secara sistematis, sehingga rute yang dihasilkan merepresentasikan jumlah sisi minimum antara titik asal dan tujuan [5]. Penelitian sebelumnya menunjukkan bahwa BFS unggul dalam hal kelengkapan eksplorasi dan konsistensi dalam menemukan solusi optimal, meskipun memerlukan penggunaan memori yang lebih besar dibandingkan algoritma lainnya [18].

Ilustrasi di bawah ini menggambarkan bagaimana BFS menelusuri simpul-simpul berdasarkan kedalaman level secara menyeluruh. Simpul A sebagai simpul awal akan menjelajahi simpul B, C, dan D terlebih dahulu, sebelum melanjutkan ke E, F, G, dan seterusnya. Panah dalam gambar menunjukkan urutan penelusurannya.




Gambar 1. Ilustrasi Proses Penelusuran *Breadth-First Search* (BFS)

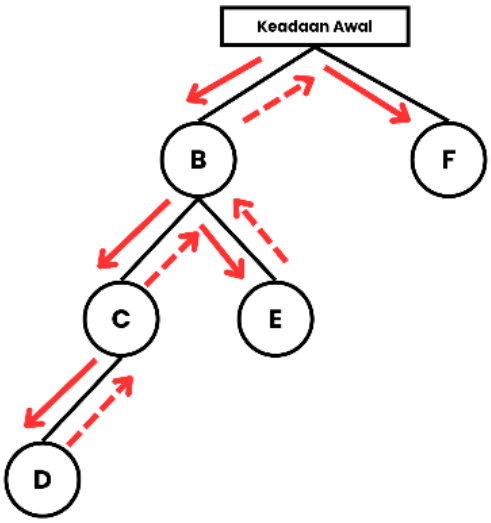
Pada penelitian ini, algoritma BFS memiliki relevansi penting untuk sistem pengiriman paket di wilayah Kaliuntu, Kabupaten Buleleng. Wilayah ini ditandai oleh jaringan jalan yang padat dengan banyak gang sempit serta simpul yang saling terhubung, sehingga pemilihan jalur tercepat menjadi kebutuhan krusial bagi kurir. Dengan merepresentasikan jaringan jalan sebagai graf tak berbobot, penerapan BFS memungkinkan sistem menemukan jalur dengan jumlah simpul minimum antara titik

asal dan tujuan secara efisien sehingga memungkinkan peningkatan efisiensi pengiriman, baik dari segi waktu maupun biaya operasional [5], [18].

### 2.6 *Depth-First Search*

*Depth-First Search* (DFS) merupakan algoritma penelusuran graf yang bekerja dengan strategi eksplorasi mendalam terlebih dahulu. Algoritma ini menggunakan struktur data *stack* atau pendekatan rekursif dengan prinsip *Last In First Out* (LIFO), dimana simpul yang terakhir dimasukkan akan diproses pertama kali [9]. Ciri khas utama DFS adalah adanya proses *backtracking*, yaitu kembali ke simpul sebelumnya ketika tidak ada lagi jalur yang dapat dieksplorasi dari simpul tertentu [18]. Dengan cara ini, DFS menelusuri graf secara mendalam pada satu cabang sebelum berpindah ke cabang lainnya, berbeda dengan BFS yang menelusuri secara menyeluruh *level-by-level*. Penelitian lain juga menunjukkan bahwa DFS memiliki keunggulan dalam hal kecepatan eksekusi dan penggunaan memori yang lebih rendah, meskipun tidak selalu menjamin jalur terpendek [5], [18].

Ilustrasi di bawah ini memperlihatkan alur penelusuran DFS pada sebuah graf. Tanda panah merah menandakan urutan kunjungan simpul dari atas ke bawah. Simpul A sebagai simpul awal akan mengeksplorasi jalur ke B, lanjut ke C, dan terus ke D. Setelah mencapai simpul D sebagai simpul ujung tanpa tetangga yang belum dikunjungi, DFS akan mundur ke simpul sebelumnya untuk mengeksplorasi jalur ke simpul lainnya seperti E hasil identifikasi dari simpul B dan seterusnya.




Gambar 2. Ilustrasi Proses Penelusuran *Depth-First Search* (DFS)

Dalam konteks pengiriman paket di wilayah Kaliuntu, DFS cenderung menyelesaikan penelusuran pada satu kawasan atau jalur sempit hingga tuntas sebelum beralih ke kawasan lain. Strategi ini berguna untuk menjelajahi seluruh kemungkinan jalur yang tersedia, terutama ketika tujuan pencarian bukan hanya mencari jarak terpendek, melainkan jalur berdasarkan kondisi atau preferensi tertentu seperti area perumahan atau *cluster* tertentu [9]. Pendekatan ini berpotensi meminimalkan perpindahan antarwilayah yang acak sehingga dapat menghemat tenaga serta waktu distribusi dalam operasional kurir ekspedisi.

### 2.7 Python

Python merupakan bahasa pemrograman tingkat tinggi yang banyak digunakan dalam penelitian ilmiah dan komputasi modern karena kesederhanaan sintaksis, fleksibilitas, serta dukungan ekosistem pustaka ilmiah yang luas. Python adalah alat komputasi penting dalam konteks penelitian berbasis algoritma dan teori graf karena mendukung proses implementasi, simulasi, dan validasi teoretis model yang dikembangkan. Berbagai studi menunjukkan bahwa Python efektif digunakan dalam

menyelesaikan permasalahan matematis dan analisis struktural graf yang kompleks [20], [21]. Dengan demikian, Python berfungsi sebagai lingkungan komputasi utama yang memungkinkan peneliti untuk secara langsung menguji dan membandingkan kinerja algoritma *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS) secara efisien.

Selain itu, kemampuan analitis ekosistem Python sangat mendukung pemodelan keterhubungan data yang relevan dengan optimasi penelusuran rute, analisis perilaku sistem kompleks, serta segmentasi data spasial di lapangan [13], [22]. Kombinasi antara kemampuan analitis dan kemudahan pemrograman ini menjadikan Python sebagai platform ideal untuk mendukung penelitian algoritmik, khususnya dalam analisis perbandingan kompleksitas waktu BFS dan DFS pada sistem penentuan rute kurir ekspedisi di Kaliuntu.

## 3. METODE PENELITIAN

### 3.1 Studi Literatur

Studi literatur dilakukan untuk membangun fondasi teoretis yang mendalam mengenai algoritma penelusuran graf, teori kompleksitas waktu, dan aplikasi praktis dalam sistem logistik. Penelusuran literatur mencakup jurnal ilmiah internasional dan nasional, tesis, serta publikasi dari berbagai *database* akademik terakreditasi. Kriteria pemilihan literatur mencakup publikasi dari tahun 2021 hingga 2025 yang secara langsung atau tidak langsung membahas perbandingan algoritma pencarian graf, analisis kompleksitas waktu, atau aplikasi algoritma dalam sistem transportasi dan logistik.

Referensi dengan metodologi yang ketat dan hasil yang terukur diprioritaskan untuk memastikan validitas dasar teoretis penelitian. Hasil studi literatur menunjukkan bahwa *Breadth-First Search* (BFS) lebih efisien dalam menemukan jalur terpendek pada graf tak berbobot dengan kompleksitas waktu $O(|V| + |E|)$, sementara *Depth-First Search* (DFS) menawarkan fleksibilitas dalam eksplorasi jalur alternatif meskipun tidak menjamin solusi optimal. Kombinasi kedua pendekatan ini memberikan perspektif yang seimbang dalam mengevaluasi algoritma pencarian rute untuk aplikasi logistik di wilayah dengan struktur jalan kompleks.

### 3.2 Pengumpulan Data

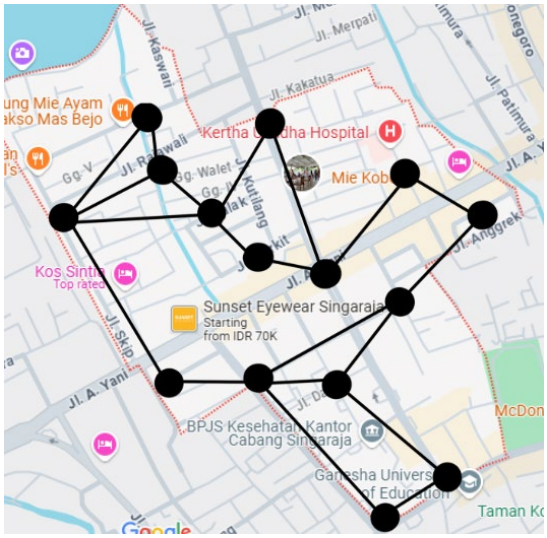
Pengumpulan data merupakan tahap fundamental dalam penelitian ini yang bertujuan memperoleh representasi akurat jaringan jalan di wilayah Kaliuntu. Data dikumpulkan melalui pendekatan triangulasi yang menggabungkan observasi lapangan langsung dengan pemetaan digital berbasis teknologi geospasial. Kombinasi kedua metode ini memastikan validitas dan reliabilitas data, sekaligus meminimalkan bias yang mungkin muncul dari penggunaan satu sumber data tunggal. Tahapan pengumpulan data ini menghasilkan informasi mengenai lokasi strategis, keterhubungan antar titik, serta karakteristik geografis wilayah yang menjadi dasar konstruksi graf untuk implementasi algoritma penelusuran.

#### 3.2.1 Observasi Lapangan

Observasi lapangan dilakukan di wilayah Kaliuntu, Kabupaten Buleleng, Bali dari tanggal 10 hingga 14 April 2025. Pengamatan fokus pada identifikasi titik-titik strategis sepanjang jaringan jalan yang dilalui oleh kurir ekspedisi. Kriteria untuk menentukan suatu lokasi sebagai simpul mencakup persimpangan jalan yang menghubungkan minimal dua jalur berbeda, lokasi penting seperti pusat layanan atau area permukiman padat yang menjadi tujuan pengiriman, serta jalur utama yang menghubungkan berbagai kawasan. Lalu dicatat nama jalan dan konektivitas setiap lokasi terhadap lokasi lainnya. Data observasi diverifikasi melalui konfirmasi dengan kurir ekspedisi yang bekerja di wilayah tersebut untuk memastikan akurasi keterhubungan antar simpul. Hasil observasi lapangan menghasilkan daftar 15 titik lokasi strategis yang akan dijadikan simpul dalam representasi graf.

### 3.2.2 Pemetaan Digital

Pemetaan digital dilakukan menggunakan *Google Maps* sebagai sumber referensi untuk memvalidasi data observasi lapangan dan mengidentifikasi rute konektivitas yang mungkin terlewatkan dalam observasi langsung. *Google Maps* menyediakan visualisasi jelas mengenai jaringan jalan, keterhubungan geografis antar titik, dan nama jalan resmi. Hasil observasi lapangan dibandingkan dengan data *Google Maps* guna menjamin akurasi dan kelengkapan informasi pada graf. Hasil dari kedua metode pengumpulan data ini menghasilkan visualisasi geografis wilayah Kaliuntu yang menunjukkan 15 titik strategis beserta keterhubungannya. Visualisasi ini ditampilkan pada Gambar 3, yang merupakan hasil pemetaan dari *Google Maps* dengan penandaan titik-titik lokasi penting dan garis penghubung antar lokasi berdasarkan hasil observasi lapangan. Pada tahap ini, simpul masih merepresentasikan kondisi geografis aktual wilayah penelitian.




Gambar 3. Pemetaan Wilayah Kaliuntu sebagai Graf Titik Pengantaran Kurir

Setelah visualisasi geografis diperoleh, setiap titik strategis kemudian diidentifikasi dengan nama jalan resmi dan dikonversi menjadi label abjad dari (A hingga O) untuk memudahkan implementasi dalam pemrograman. Konektivitas antar simpul ditentukan berdasarkan keterhubungan langsung melalui jalur jalan, menghasilkan total 20 sisi dalam representasi graf tak berarah. Lalu Tabel 1 menampilkan daftar lengkap simpul beserta nama jalan yang akan direpresentasikan.

Tabel 1. Representasi Data Simpul Titik Pengantaran Kurir

| Simpul | Nama Jalan |
|---|---|
| A | Jl. Cendrawasih |
| B | Jl. Kenanga |
| C | Jl. Dewi Sartika |
| D | Jl. A. Yani |
| E | Jl. Tasbih |
| F | Jl. Jatayu |
| G | Jl. Rajawali |
| H | Jl. Jalak |
| I | Jl. Parkit |
| J | Jl. Dahlia |
| K | Jl. Udayana |

| | |
|---|---|
| L | Jl. Angsoka |
| M | Jl. Kartini |
| N | Jl. Skip |
| O | Jl. Mawar |

Proses transformasi dataset dari data mentah jaringan nyata menjadi data yang siap diolah disajikan dalam bentuk diagram alir pada Gambar 4. Tahapan ini dimulai dari tahap Pengumpulan Data Mentah (observasi lapangan dan Google Maps), dilanjutkan dengan tahap Identifikasi Titik Strategis (penentuan titik persimpangan dan jalur konektivitas), lalu tahap Pelabelan Titik Lokasi (konversi titik menjadi simpul A-O). Selanjutnya, data masuk ke tahap Pemodelan Struktur Graf (representasi menjadi graf tak berarah dan tak berbobot), hingga akhirnya masuk pada tahap Penyusunan Data Siap Olah berupa daftar ketetanggaan (*adjacency list*) yang akan diproses langsung oleh program.

Gambar 4. Diagram Alur Transformasi Dataset

Melalui proses abstraksi ini, elemen-elemen fisik dunia nyata seperti kondisi jalan aspal yang berliku, jarak tempuh dalam satuan meter, hingga potensi hambatan kemacetan dihilangkan sepenuhnya (diabstraksikan). Sisa dari proses abstraksi tersebut hanyalah relasi murni antar persimpangan yang didefinisikan secara struktural, sehingga algoritma Python dapat membaca dan mengeksekusi penelusuran secara optimal berdasarkan konektivitas logis. Tahap representasi graf secara formal ini sangat bergantung pada data yang diperoleh dari proses observasi lapangan dan pemetaan digital ini. Untuk memastikan efisiensi algoritma penelusuran yang akan digunakan, metode representasi yang tepat diperlukan untuk mengubah data geografis mentah ke dalam struktur graf yang dapat diproses secara komputasional.

### 3.3 Representasi Graf dan Struktur Data

Sebelum implementasi algoritma dilakukan, graf hasil pemetaan wilayah direpresentasikan dalam bentuk yang sesuai untuk pemrograman. Penelitian ini mengadopsi representasi berbasis *adjacency list*, dengan setiap simpul menyimpan daftar simpul tetangga yang terhubung secara langsung. Sebagaimana telah dipaparkan pada bagian *adjacency list*, pilihan ini didasarkan pada efisiensi ruang penyimpanan $O(|V| + |E|)$ yang lebih optimal dibandingkan *adjacency matrix* dengan kompleksitas $O(|V|^2)$, terutama karena jaringan jalan Kaliuntu merupakan sparse graph dengan jumlah sisi jauh lebih kecil daripada $|V|^2$.

Representasi graf formal yang digunakan adalah $G = (V, E)$, dengan $V$ merupakan himpunan simpul yang terdiri dari 15 elemen berlabel (A hingga O), dan $E$ merupakan himpunan sisi yang menghubungkan simpul berdasarkan konektivitas jalan. Setiap sisi dalam graf bersifat tak berarah (*undirected*), artinya koneksi antara dua simpul dapat dilalui dari kedua arah, dan tak berbobot (*unweighted*), sehingga setiap sisi memiliki nilai yang sama tanpa mempertimbangkan jarak atau waktu tempuh aktual.

Pada tahap implementasi, *adjacency list* direpresentasikan menggunakan *dictionary* dalam Python, dengan setiap kunci merupakan label simpul dan nilai berupa list dari simpul-simpul tetangga yang terhubung langsung. Struktur data *dictionary* dipilih karena memungkinkan akses $O(1)$ untuk mengecek keberadaan simpul tertentu dan $O(degree)$ untuk melakukan iterasi terhadap semua tetangga dari suatu simpul. Efisiensi akses ini mendukung implementasi algoritma BFS dan DFS yang memerlukan penelusuran tetangga secara berulang selama proses penelusuran berlangsung.

### 3.4 Implementasi Algoritma dan *Pseudocode*

Setelah struktur graf dan representasi data ditetapkan, tahap selanjutnya adalah implementasi algoritma BFS dan DFS ke dalam bahasa pemrograman. Proses implementasi dimulai dengan penyusunan *pseudocode* sebagai *blueprint* logika algoritma, dilanjutkan dengan penerjemahan ke dalam kode Python yang dapat dieksekusi. Implementasi dirancang sedemikian rupa untuk memastikan pengukuran waktu eksekusi yang akurat dan memungkinkan variasi rute melalui mekanisme randomisasi. Sub-bab ini menjelaskan secara rinci langkah-langkah implementasi mulai dari persiapan *pseudocode* hingga modifikasi khusus untuk keperluan eksperimen.

Sebelum tahap implementasi ke dalam bahasa pemrograman, kedua algoritma terlebih dahulu dituangkan ke dalam bentuk *pseudocode*. Langkah ini sangat penting untuk memastikan bahwa logika fundamental dari masing-masing algoritma dapat dipahami dengan baik, divalidasi secara teoretis, dan direplikasi secara akurat. *Pseudocode* ini berfungsi sebagai cetak biru (*blueprint*) yang menjembatani konsep matematis graf dengan sintaksis pemrograman. Selain itu, *pseudocode* juga menjadi acuan utama dalam menyusun kode program sekaligus membantu peneliti mengidentifikasi poin-poin kritis yang berpotensi memengaruhi pengukuran waktu eksekusi. Pada tahap pertama, algoritma BFS diimplementasikan dengan memanfaatkan struktur data antrean (*queue*) yang beroperasi berdasarkan prinsip FIFO. *Pseudocode* algoritma BFS yang digunakan dalam penelitian ini adalah sebagai berikut:

Algoritma 1. *Breadth-First Search* (BFS)

**Require:** Graph G = (V, E), starting vertex s ∈ V
**Ensure:** route, a list of vertices visited in BFS order, covering all vertices in V

```
1:  procedure BFS(G, s)
2:      for each vertex v in V do
3:          visited[v] ← false
4:      end for
5:      create empty queue Q
6:      create empty list route
7:      enqueue(Q, s)
8:      visited[s] ← true
9:
10:     while Q is not empty do
11:         v ← dequeue(Q)              # Remove from front of queue
12:         route.append(v)             # Add to traversal result
13:         for each neighbor w of v do
14:             if not visited[w] then
15:                 visited[w] ← true
16:                 enqueue(Q, w)       # Add to back of queue
17:             end if
18:         end for
19:     end while
20:
21:     return route
22: end procedure
```

*Pseudocode* tersebut menunjukkan alur eksekusi BFS secara sistematis dalam menangani struktur *adjacency list*. Algoritma ini selalu dimulai dengan menginisialisasi dan menandai semua simpul sebagai entitas yang belum dikunjungi, kemudian memasukkan simpul awal ke dalam antrean. Proses komputasi iteratif kemudian dilakukan di mana simpul dikeluarkan dari bagian depan antrean, ditambahkan ke dalam daftar rute hasil, dan semua tetangganya yang belum dikunjungi ditandai serta

dimasukkan ke bagian belakang antrean. Strategi FIFO ini secara mutlak memastikan bahwa simpul yang berada lebih dekat (berdasarkan jumlah sisi dari titik awal) selalu diproses terlebih dahulu, sehingga menciptakan pola penelusuran yang melebar (*level-by-level*).

Sebagai perbandingan dari pendekatan BFS, algoritma DFS diimplementasikan menggunakan logika yang berbeda secara fundamental. DFS menggunakan struktur data tumpukan (*stack*) dengan prinsip LIFO melalui pendekatan rekursif yang memanfaatkan *call stack* pada sistem secara implisit. Pendekatan ini memungkinkan penelusuran graf bergerak sedalam mungkin ke satu rute spesifik sebelum akhirnya melakukan langkah mundur. *Pseudocode* algoritma DFS yang digunakan dalam penelitan ini adalah sebagai berikut:

Algoritma 2. *Depth-First Search* (DFS)

**Require:** Graph G = (V, E), starting vertex s ∈ V
**Ensure:** route, a list of vertices visited in DFS order, covering all vertices in V

```
1:  procedure DFS(G, s)
2:      for each vertex v in V do
3:          visited[v] ← false
4:      end for
5:      create empty stack S
6:      create empty list route
7:      push(S, s)
8:      visited[s] ← true
9:      route.append(s)
10:
11:     while S is not empty do
12:         current ← top(S)                    # Peek without popping
13:         found_unvisited ← false
14:
15:         for each neighbor w of current do
16:             if not visited[w] then
17:                 push(S, w)
18:                 visited[w] ← true
19:                 route.append(w)
20:                 found_unvisited ← true
21:                 break                       # Continue exploring depth
22:             end if
23:         end for
24:
25:         if not found_unvisited then
26:             pop(S)                          # Backtrack
27:         end if
28:     end while
29:
30:     return route
31: end procedure
```

*Pseudocode* DFS iteratif ini menggunakan *stack* eksplisit untuk menyimpan simpul yang sedang dieksplorasi. Pada setiap iterasi, simpul teratas *stack* diperiksa tanpa dikeluarkan, kemudian algoritma mencari tetangga pertama yang belum dikunjungi. Jika ditemukan, tetangga tersebut ditambahkan ke *stack* untuk eksplorasi lebih dalam. Jika tidak ada tetangga yang belum dikunjungi, simpul dikeluarkan

dari *stack* untuk melakukan *backtrack* ke simpul sebelumnya. Pendekatan iteratif ini dipilih untuk menghindari keterbatasan kedalaman rekursi Python dan memberikan kontrol yang lebih eksplisit terhadap proses *backtracking*.

### 3.4.1 Implementasi dalam Bahasa Pemrograman Python

Implementasi kedua algoritma dilakukan dalam bahasa pemrograman Python 3.10 dengan menggunakan *library* standar *time* untuk pengukuran waktu eksekusi, *random* untuk randomisasi urutan eksplorasi tetangga, dan *collections.deque* untuk implementasi *queue* pada BFS. Graf direpresentasikan sebagai *adjacency list* menggunakan *dictionary* Python dengan *format key-value pair* yaitu kunci label simpul dan nilai adalah list simpul tetangga.

BFS diimplementasikan menggunakan struktur data *queue* dengan prinsip FIFO melalui *collections.deque*, sementara DFS menggunakan pendekatan iteratif dengan *stack* eksplisit berupa *list* Python. Kedua algoritma mencatat waktu eksekusi dan mengembalikan rute penelusuran lengkap beserta waktu eksekusi dalam milidetik. Implementasi mengikuti *pseudocode* standar dengan satu modifikasi untuk keperluan eksperimen, yaitu penambahan *random.shuffle()* sebelum iterasi tetangga yang menciptakan variasi rute non-deterministik untuk menguji stabilitas performa algoritma terhadap berbagai skenario penelusuran.

## 3.5 Eksperimen dan Pengujian

Eksperimen dirancang untuk menghasilkan data empiris yang valid dan reliabel mengenai performa algoritma BFS dan DFS dalam penelusuran graf jaringan jalan Kaliuntu. Pengujian dilakukan dalam lingkungan komputasi yang terkontrol dengan protokol pengukuran yang ketat untuk meminimalkan bias dan variabilitas eksternal. Setiap algoritma diuji sebanyak 10 kali iterasi dengan struktur graf dan simpul awal yang identik untuk mengamati konsistensi performa.

Simpul awal ditetapkan sebagai simpul A yang merepresentasikan pusat ekspedisi atau gudang utama, dengan tujuan penelusuran mengunjungi semua 15 simpul dalam graf. Kriteria penghentian adalah semua simpul yang terjangkau telah dikunjungi atau tidak ada lagi simpul yang dapat diakses dari simpul terakhir. Kurir tidak diwajibkan kembali ke simpul awal setelah seluruh simpul dikunjungi, mencerminkan kondisi nyata pengiriman paket seorang kurir yang memiliki fleksibilitas rute dan tidak selalu kembali ke titik awal setelah menyelesaikan pengantaran. Graf direpresentasikan dengan adjacency list yang sama tanpa modifikasi struktur pada semua pengujian untuk memastikan konsistensi kondisi eksperimen.

### 3.5.1 Lingkungan Pengujian

Pengujian dilaksanakan pada lingkungan interaktif menggunakan Jupyter Notebook yang memungkinkan eksekusi kode secara berulang, visualisasi hasil langsung, dan pencatatan waktu eksekusi dengan presisi tinggi. Spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam pengujian ditampilkan pada Tabel 2 untuk memastikan transparansi dan replikabilitas hasil penelitian.

Tabel 2. Spesifikasi Perangkat Keras dan Perangkat Lunak

| Komponen | Spesifikasi |
|---|---|
| Perangkat | Laptop Acer E5-475G |
| Prosesor | Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz |
| RAM | 8 GB |
| Sistem Operasi | Windows 10 |
| Lingkungan Eksekusi | Jupyter Notebook |
| Bahasa Pemrograman | Python 3.10 |

### 3.5.2 Protokol Pengukuran Waktu

Pengukuran waktu eksekusi dilakukan menggunakan fungsi *time.time()* dari modul standar Python yang mengembalikan *timestamp* dengan presisi milidetik. Protokol pengukuran diintegrasikan langsung dalam fungsi *traversal* melalui serangkaian langkah yang berurutan. Proses ini dimulai dengan pencatatan waktu awal sebelum eksekusi *loop traversal* menggunakan perintah *start_time = time.time()*. Setelah itu, algoritma dieksekusi secara penuh hingga seluruh simpul berhasil dikunjungi berawal dari simpul A. Tahapan selanjutnya adalah pencatatan waktu akhir setelah *loop* selesai menggunakan perintah *end_time = time.time()*. Terakhir, perhitungan waktu eksekusi dilakukan dengan mencari selisih antara waktu akhir dan waktu awal yang kemudian dikalikan 1000 untuk mengonversinya ke dalam satuan milidetik.

Penelitian ini menggunakan *single-run measurement* per eksperimen dengan total 10 eksperimen berulang untuk masing-masing algoritma. Pendekatan ini dipilih karena setiap eksperimen sudah menghasilkan variasi rute akibat *random.shuffle()*, sehingga 10 eksperimen independen lebih representatif untuk mengukur stabilitas algoritma dibandingkan *averaging*. Seluruh eksperimen dijalankan dalam sesi tunggal tanpa aplikasi background untuk meminimalkan *noise* dari *interrupt* sistem, *context switching*, atau *cache miss* yang dapat memengaruhi konsistensi pengukuran.

## 3.6 Analisis Data

Selanjutnya adalah tahap analisis data yang bertujuan untuk membandingkan performa algoritma BFS dan DFS berdasarkan hasil eksperimen yang telah dilakukan. Analisis meliputi pengukuran efisiensi waktu eksekusi, evaluasi konsistensi rute penelusuran, serta observasi pola perilaku penelusuran kedua algoritma pada graf wilayah Kaliuntu. Proses analisis menggunakan kombinasi metode komputasional untuk perhitungan statistik dan juga metode observasional untuk mengidentifikasi pola kualitatif.

Secara lebih rinci, analisis data melibatkan empat metrik utama guna memberikan perbandingan komprehensif antara algoritma BFS dan DFS baik dari segi kuantitatif maupun kualitatif. Metrik pertama adalah efisiensi waktu yang dievaluasi berdasarkan statistik deskriptif dari 10 kali pengujian tiap algoritma mencakup nilai rata-rata, minimum, dan maksimum untuk melihat variasi kinerja. Metrik kedua berupa rasio performa yang dihitung dengan membandingkan total waktu eksekusi DFS terhadap BFS, di mana nilai rasio di atas 1 berarti DFS lebih lambat sedangkan di bawah 1 menunjukkan DFS lebih cepat. Selanjutnya, metrik ketiga mengukur variasi dan konsistensi rute melalui jumlah rute unik hasil 10 pengujian tiap algoritma. Rute unik yang lebih sedikit menandakan konsistensi tinggi, sedangkan jumlah rute yang lebih banyak menunjukkan tingginya fleksibilitas penelusuran. Metrik terakhir menganalisis pola penelusuran secara kualitatif melalui urutan kunjungan simpul, pola spasial antar *cluster*, frekuensi *backtracking* pada DFS, serta kesesuaian pergerakan terhadap topografi wilayah Kaliuntu.

Data dianalisis dengan menggunakan *tools* pemrograman Python dan pengamatan manual. Fungsi *analyze_results()* yang dikembangkan secara khusus digunakan untuk mengekstraksi waktu eksekusi dari setiap hasil pengujian, melakukan perhitungan statistik deskriptif menggunakan fungsi yang sudah ada dalam Python, menemukan rute unik dengan mengkonversi ke *tuple* dan menyimpannya dalam set, dan menghitung rasio performa. Untuk membuat tabel komparatif dan menampilkan grafik perbandingan, hasil eksperimen ditransfer secara manual ke Microsoft Excel dari *console output Jupyter Notebook*. Observasi pola rute penelusuran dilakukan secara manual dengan membandingkan output rute dari setiap eksperimen untuk memeriksa pola spasial, *cluster* geografis, dan transisi antar *cluster*.

## 3.7 Penarikan Kesimpulan

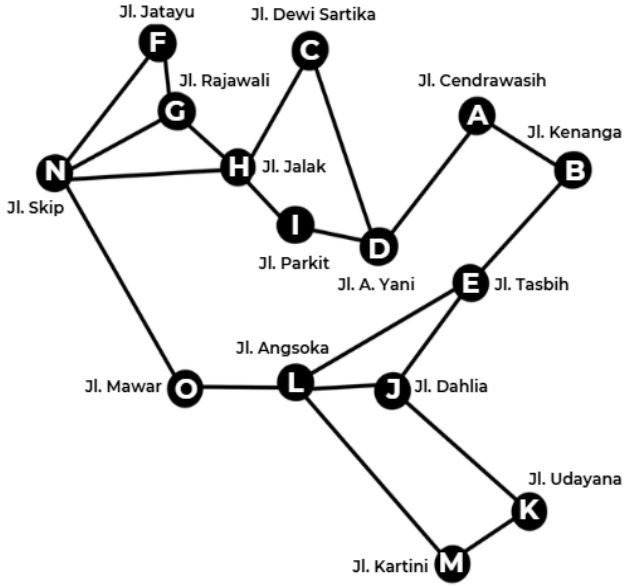
Kesimpulan penelitian ditarik berdasarkan hasil analisis data eksperimen dengan mempertimbangkan beberapa aspek yang saling melengkapi untuk memperoleh pemahaman komprehensif mengenai kinerja algoritma *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS) dalam konteks sistem penentuan rute kurir di wilayah Kaliuntu. Aspek pertama menilai efisiensi waktu eksekusi dari rata-rata waktu komputasi beserta rasio hasil pengujian. Pada evaluasi ini, algoritma

dengan waktu rata-rata terendah dan variasi kecil dianggap paling efisien sekaligus stabil. Aspek kedua mengevaluasi keandalan dan konsistensi dari kemampuan algoritma menghasilkan rute dengan variasi minimal. Semakin sedikit variasi rute yang dihasilkan, maka semakin tinggi konsistensi dan prediktabilitas algoritma tersebut dalam operasional berulang. Aspek ketiga meninjau kesesuaian kontekstual yang berfokus pada relevansi karakteristik algoritma terhadap kondisi nyata di Kaliuntu seperti kepadatan jalan, pola permukiman, dan efisiensi spasial. Penilaian pada aspek ini turut mencakup kemampuan strategi penelusuran dalam menyesuaikan diri dengan kompleksitas jaringan jalan lokal.

## 4. HASIL DAN PEMBAHASAN

### 4.1 Representasi Graf

Graf yang digunakan dalam penelitian ini terdiri dari 15 simpul yang diperoleh dari pengumpulan data pemetaan wilayah Kaliuntu, Bali. Setiap simpul merepresentasikan titik strategis jalanan utama, dengan label abjad (A hingga O). Representasi graf ini bersifat tidak berarah dan tidak berbobot, dengan koneksi antar simpul menggambarkan keterhubungan langsung antar persimpangan jalan. Fokus difokuskan pada urutan penelusuran simpul tanpa memperhitungkan jarak tempuh aktual.




Gambar 5. Representasi Titik Pengantaran Kurir sebagai Graf Tidak Berbobot

Visualisasi struktur graf ini menunjukkan distribusi geografis simpul dan keterhubungannya yang mencerminkan kondisi nyata di lapangan. Struktur ini direpresentasikan dalam bentuk *adjacency list* untuk memudahkan implementasi algoritma pemrograman. Tabel 3 menunjukkan daftar ketetanggaan (*adjacency list*) tersebut.

Tabel 3. Daftar Ketetanggaan Simpul (*Adjacency List*) Graf Tidak Berbobot

| Simpul | Daftar Simpul Tetangga |
|---|---|
| A | B, D |
| B | A, E |
| C | D, H |
| D | A, C, I |
| E | B, J, L |

| | |
|---|---|
| F | G, N |
| G | F, H, N |
| H | C, G, I, N |
| I | D, H |
| J | E, K, L |
| K | J, M |
| L | E, J, O |
| M | K, L |
| N | F, G, H, O |
| O | L, N |

### 4.2 Hasil Implementasi Algoritma

Implementasi algoritma BFS dan DFS dilakukan menggunakan Python dengan simpul A ditetapkan sebagai titik awal penelusuran (pusat ekspedisi). Masing-masing algoritma diuji sebanyak 10 kali iterasi untuk mengukur performa dan variasi rute secara konsisten. Berikut merupakan hasil eksperimen dan pengujian untuk masing-masing algoritma:

Tabel 4. Hasil Eksperimen Penelusuran *Breadth-First Search* (BFS)

| Eksperimen ke- | Rute Penelusuran | Waktu Eksekusi |
|---|---|---|
| 1 | A→ D→ B→ I→ C→ E→ H→ L→ J→ N→ G→ O→ K→ F→ M | 0.072 ms |
| 2 | A→ B→ D→ E→ C→ I→ L→ J→ H→ O→ K→ N→ G→ M→ F | 0.043 ms |
| 3 | A→ D→ B→ I→ C→ E→ H→ L→ J→ N→ G→ O→ K→ F→ M | 0.038 ms |
| 4 | A→ B→ D→ E→ C→ I→ J→ L→ H→ K→ O→ N→ G→ M→ F | 0.108 ms |
| 5 | A→ B→ D→ E→ C→ I→ L→ J→ H→ O→ K→ G→ N→ M→ F | 0.093 ms |
| 6 | A→ D→ B→ C→ I→ E→ H→ L→ J→ G→ N→ O→ K→ F→ M | 0.095 ms |
| 7 | A→ B→ D→ E→ C→ I→ J→ L→ H→ K→ O→ N→ G→ M→ F | 0.093 ms |
| 8 | A→ D→ B→ C→ I→ E→ H→ L→ J→ G→ N→ O→ K→ F→ M | 0.090 ms |
| 9 | A→ B→ D→ E→ C→ I→ J→ L→ H→ K→ O→ G→ N→ M→ F | 0.088 ms |
| 10 | A→ B→ D→ E→ C→ I→ L→ J→ H→ O→ K→ G→ N→ M→ F | 0.097 ms |

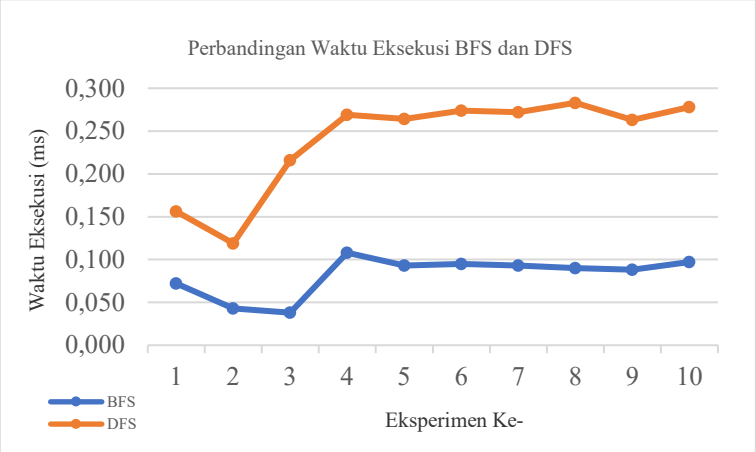
Tabel 5. Hasil Eksperimen Penelusuran *Depth-First Search* (DFS)

| Eksperimen ke- | Rute Penelusuran | Waktu Eksekusi |
|---|---|---|
| 1 | A→ B→ E→ J→ K→ M→ L→ O→ N→ G→ H→ C→ D→ I→ F | 0.084 ms |
| 2 | A→ B→ E→ L→ J→ K→ M→ O→ N→ G→ H→ C→ D→ I→ F | 0.076 ms |
| 3 | A→ B→ E→ L→ O→ N→ G→ F→ H→ C→ D→ I→ J→ K→ M | 0.178 ms |
| 4 | A→ B→ E→ L→ J→ K→ M→ O→ N→ G→ F→ H→ C→ D→ I | 0.161 ms |
| 5 | A→ D→ I→ H→ N→ G→ F→ O→ L→ E→ J→ K→ M→ B→ C | 0.171 ms |
| 6 | A→ D→ C→ H→ G→ N→ O→ L→ E→ B→ J→ K→ M→ F→ I | 0.179 ms |
| 7 | A→ D→ C→ H→ N→ F→ G→ O→ L→ J→ E→ B→ K→ M→ I | 0.179 ms |
| 8 | A→ D→ I→ H→ G→ N→ F→ O→ L→ J→ K→ M→ E→ B→ C | 0.193 ms |
| 9 | A→ B→ E→ L→ J→ K→ M→ O→ N→ G→ H→ C→ D→ I→ F | 0.175 ms |
| 10 | A→ D→ I→ H→ C→ G→ N→ F→ O→ L→ E→ B→ J→ K→ M | 0.181 ms |

Dari 10 eksperimen, BFS menghasilkan 6 rute unik dengan pola penelusuran yang konsisten dan terstruktur. Karakteristik penelusurannya cenderung menjelajahi simpul terdekat sebelum melanjutkan ke level berikutnya. Sebaliknya, DFS menghasilkan 9 rute unik, menunjukkan fleksibilitas penelusuran yang lebih tinggi dengan menghasilkan variasi rute yang lebih beragam meskipun beroperasi pada graf yang sama.

### 4.3 Perbandingan Kompleksitas Waktu

Dari hasil implementasi algoritma menunjukkan bahwa waktu eksekusi algoritma BFS dan DFS memiliki perbedaan yang mencerminkan karakteristik algoritma masing-masing. Grafik berikut memperlihatkan fluktuasi waktu eksekusi BFS dan DFS selama 10 iterasi:




Gambar 6. Grafik Perbandingan Waktu Eksekusi BFS dan DFS

Berdasarkan Gambar 6, terlihat secara visual bahwa garis waktu eksekusi BFS secara konsisten berada di bawah garis eksekusi DFS pada setiap iterasi. Fluktuasi waktu yang terjadi pada kedua algoritma tergolong stabil dan tidak menunjukkan lonjakan anomali yang signifikan. Untuk merinci sebaran nilai dari visualisasi tersebut, analisis statistik deskriptif mengenai nilai minimum, rata-rata, dan maksimum dari waktu eksekusi kedua algoritma dirangkum secara komprehensif pada Tabel 6.

Tabel 6. Analisis Perbandingan Waktu Eksekusi BFS dan DFS

| Algoritma | Waktu minimal | Waktu rata-rata | Waktu maksimal |
|---|---|---|---|
| BFS | 0.038ms | 0.082ms | 0.108ms |
| DFS | 0.076ms | 0.158ms | 0.193ms |

Perhitungan rasio antara rata-rata waktu eksekusi algoritma DFS terhadap BFS menghasilkan:

$$Rasio = \frac{Waktu\ rata-rata\ DFS}{Waktu\ rata-rata\ BFS} = \frac{0.158}{0.082} \approx 1.93$$

Nilai aproksimasi ini menunjukkan bahwa DFS membutuhkan waktu hampir dua kali lebih lama dibandingkan BFS dalam menyelesaikan penelusuran pada graf hasil pemetaan wilayah.

### 4.4 Evaluasi Performa Empiris terhadap Kompleksitas Teoretis

Penting untuk ditegaskan bahwa pengukuran waktu eksekusi (*running time*) dalam satuan milidetik (ms) pada penelitian ini bukanlah indikator empiris mutlak atas performa kecepatan algoritma. Secara matematis dan teoretis, baik BFS maupun DFS memiliki kompleksitas waktu yang sama persis,

yaitu $O(V + E)$. Waktu eksekusi dalam penelitian ini dikaji murni sebagai instrumen pembanding terhadap kompleksitas teoretis tersebut dalam lingkungan komputasi yang seragam. Perlu dicatat pula sebagai batasan penelitian bahwa seluruh spesifikasi eksekusi ini sangat bergantung pada konfigurasi perangkat keras dan lunak yang telah didefinisikan pada Tabel 2.

Perbedaan empiris waktu eksekusi di mana DFS tercatat hampir dua kali lebih lambat dipicu oleh tiga faktor praktis komputasi yang saling berkaitan. Faktor pertama didorong oleh karakteristik struktur graf jaringan jalan Kaliuntu yang memiliki banyak cabang buntu dan siklus tidak beraturan sehingga memaksa DFS lebih sering melakukan langkah mundur (*backtracking*). Faktor kedua dipengaruhi oleh arah penelusuran dan frekuensi pengecekan status kunjungan simpul (*visited node*) yang mencerminkan pola perlakuan komputasi berbeda antara kedua algoritma. Faktor ketiga sekaligus yang fundamental terletak pada alokasi memori *compiler* Python yang memberikan penanganan berbeda terhadap struktur data. BFS menggunakan antrean FIFO yang dioptimasi pada elemen terdepan, sementara DFS mengandalkan tumpukan LIFO yang secara konstan memanipulasi ujung indeks memori sehingga menciptakan *overhead* waktu tambahan pada sistem.

### 4.5 Analisis Pola Traversal dan Interpretasi Hasil

Selain perbedaan kuantitatif, fokus utama komparasi ini terletak pada perilaku penelusuran spasial (*spatial traversal pattern*) di jaringan jalan nyata. Berdasarkan urutan rute eksperimen, BFS menunjukkan pola penyebaran secara radial. Karena BFS menelusuri graf berdasarkan level (*breadth*), algoritma ini sering "melompat" antar area geografis (misalnya, dari sisi Barat Kaliuntu tiba-tiba menelusuri simpul di sisi Timur hanya karena simpul tersebut memiliki level kedalaman yang sama dari titik awal). Dalam konteks logistik nyata, pola ini sangat tidak efisien; kurir akan kelelahan karena harus bergerak bolak-balik antar klaster wilayah yang berjauhan.

Sebaliknya, meskipun mencatat waktu komputasi yang sedikit lebih lama, DFS menunjukkan pola pergerakan spasial yang jauh lebih rasional untuk operasional kurir. DFS masuk menelusuri satu jalur atau area perumahan (*cluster*) secara linier hingga ke ujung, sebelum melakukan *backtracking* dan berpindah ke area tetangga. Secara operasional di lapangan, strategi ini meminimalkan perpindahan antarwilayah yang acak, menghemat bahan bakar, energi, dan secara keseluruhan menyelaraskan logika algoritmik dengan intuisi pergerakan kurir manusia di jaringan jalan yang tidak teratur.

Meskipun BFS kurang efisien secara spasial, konsistensi rute yang dihasilkannya menjadi parameter komputasi yang patut dipertimbangkan. Dari hasil pengujian, BFS hanya menghasilkan 6 rute unik, menunjukkan tingkat prediktabilitas yang tinggi dalam pola penelusurannya. Sebaliknya, DFS menghasilkan 9 rute unik, mencerminkan fleksibilitas dan variabilitas komputasi yang lebih tinggi. Fleksibilitas ini berpotensi menjadi keuntungan besar dalam skenario yang membutuhkan eksplorasi jalur alternatif untuk menemukan solusi adaptif terhadap perubahan kondisi jalanan di lapangan.

Untuk memvalidasi stabilitas rasio performa 1.93 serta mengidentifikasi potensi perubahan perilaku algoritma seiring peningkatan beban pengujian, eksperimen lanjutan dilakukan dengan memvariasikan jumlah iterasi. Tabel 7 menyajikan data eksperimen komparatif pada skala 10, 20, 30, 50, 100, hingga 1000 iterasi.

Tabel 7. Analisis Rasio Performa pada Berbagai Jumlah Iterasi

| Jumlah Iterasi | Waktu rata-rata BFS | Waktu rata-rata DFS | Rasio DFS/BFS |
|---|---|---|---|
| 10 | 0.082ms | 0.158ms | 1.93 |
| 10 | 0.081ms | 0.156ms | 1.92 |
| 10 | 0.070ms | 0.152ms | 2.17 |
| 20 | 0.082ms | 0.191ms | 2.33 |
| 20 | 0.117ms | 0.233ms | 2.00 |
| 50 | 0.081ms | 0.184ms | 2.27 |
| 50 | 0.071ms | 0.150ms | 2.10 |

| | | | |
|---|---|---|---|
| 100 | 0.170ms | 0.416ms | 2.45 |
| 1000 | 0.072ms | 0.154ms | 2.15 |

Hasil pada Tabel 7 memberikan validasi empiris bahwa rasio performa antara DFS dan BFS tetap stabil pada kisaran angka dua, bahkan ketika beban iterasi ditingkatkan secara eksponensial hingga 1000 kali. Konsistensi pola rasio ini menegaskan bahwa perbedaan waktu eksekusi yang terjadi adalah murni akibat karakteristik fundamental algoritmik dari strategi traversal dan penanganan memori struktur data, bukan dipengaruhi oleh faktor eksternal sesaat seperti beban sistem operasi atau optimasi *compiler*.

### 4.6 Implikasi Penelitian

Hasil penelitian menunjukkan bahwa pemilihan algoritma pencarian rute pada sistem logistik tidak dapat semata-mata didasarkan pada perhitungan metrik kecepatan eksekusi program. Faktor spasial seperti konsistensi area penelusuran terbukti memiliki pengaruh yang jauh lebih vital terhadap praktik distribusi di lapangan. Meskipun secara komputasi Python BFS lebih cepat, DFS menghasilkan pola rute linier yang sejalan dengan efisiensi tenaga kerja di dunia nyata.

Dalam penelitian ini, graf dimodelkan sebagai graf tak berbobot untuk mengisolasi analisis murni pada urutan simpul. Implikasi ini membuka peluang bagi penelitian lanjutan untuk mengintegrasikan pendekatan berbasis bobot (*weighted graph*) maupun metode heuristik untuk menghasilkan sistem navigasi rute yang optimal secara komputasi sekaligus relevan secara geografis.

## 5. KESIMPULAN

Berdasarkan studi komparatif pada jaringan jalan nyata di Kaliuntu, algoritma BFS menunjukkan performa komputasi empiris yang lebih cepat dan konsisten dengan rata-rata waktu eksekusi 0.082 ms berbanding 0.158 ms milik DFS. Meskipun secara teoretis kedua algoritma memiliki kompleksitas waktu yang sama yakni $O(V + E)$, selisih kecepatan ini murni dipicu oleh kendala teknis pemrosesan sistem. Perlambatan pada DFS diakibatkan oleh penanganan memori tumpukan (LIFO) serta tingginya frekuensi *backtracking* saat merespons banyaknya siklus dan jalur buntu pada topologi graf wilayah tersebut.

Meskipun BFS unggul dalam metrik kecepatan program, analisis pola *traversal* justru mengungkap sebuah paradoks operasional yang signifikan. Penelusuran radial pada BFS memaksa pergerakan kurir melompat antar klaster wilayah secara tidak efisien, sedangkan DFS menawarkan pergerakan spasial linier yang jauh lebih rasional dengan menuntaskan pengantaran pada satu kawasan utuh sebelum berpindah ke kawasan lain. Hal ini menegaskan bahwa efisiensi logistik modern wajib menyelaraskan logika algoritmik dengan intuisi pergerakan manusia di lapangan. Algoritma BFS sangat ideal untuk navigasi jarak terpendek absolut, sementara DFS terbukti jauh lebih adaptif dan aplikatif sebagai fondasi operasional kurir ekspedisi berbasis klaster geografis.

## 6. SARAN

Berdasarkan hasil penelitian ini, terdapat beberapa usulan yang dapat diterapkan untuk pengembangan penelitian selanjutnya. Saran pertama adalah perlunya mengintegrasikan algoritma BFS dan DFS dengan algoritma heuristik seperti A* atau Dijkstra berbobot guna meningkatkan akurasi rute dengan mempertimbangkan faktor bobot jarak atau waktu tempuh aktual antar simpul. Saran kedua berfokus pada penambahan parameter kondisi lalu lintas, cuaca, dan waktu tempuh riil untuk menghasilkan simulasi yang jauh lebih mendekati situasi lapangan serta meningkatkan validitas hasil dalam konteks operasional kurir. Selain itu, pengembangan aplikasi berbasis navigasi waktu nyata (*real-time navigation*) juga sangat disarankan agar dapat diimplementasikan langsung pada perangkat kurir melalui integrasi GPS dan peta interaktif guna mendukung kelancaran pengiriman paket.

Penelitian selanjutnya juga diharapkan mampu memperluas wilayah studi ke daerah lain dengan struktur jalan yang berbeda, misalnya wilayah perkotaan padat, pedesaan, atau topografi berbukit. Perluasan wilayah ini penting untuk menguji generalisasi hasil penelitian sekaligus mengidentifikasi

karakteristik graf yang paling sesuai untuk setiap algoritma. Terakhir, para peneliti disarankan untuk melakukan analisis kompleksitas ruang (*space complexity*) sebagai pelengkap dari analisis kompleksitas waktu, sehingga temuan di masa mendatang dapat memberikan gambaran yang lebih komprehensif mengenai efisiensi kedua algoritma dalam mengelola penggunaan memori serta sumber daya komputasi.

## 7. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada para kurir ekspedisi di wilayah Kaliuntu atas kesediaannya memberikan informasi dan memvalidasi rute pengantaran secara langsung. Apresiasi juga ditujukan kepada rekan-rekan sejawat atas dukungan teknis selama proses observasi lapangan dan pemetaan digital. Diharapkan penelitian ini dapat memberikan kontribusi nyata dalam mengoptimalkan efisiensi rute pada sistem logistik modern di Indonesia.

## DAFTAR PUSTAKA

[1] I. Baidlowi and W. T. Subroto, "The Effect of E-Commerce on Economics Growth in Indonesia: Literature Review Approach," *International Research Journal of Economics and Management Studies*, vol. 3, no. 8, pp. 496–504, Aug. 2024, doi: 10.56472/25835238/irjems-v3i8p159.

[2] E. Yulianto and A. Wulandari, "Dampak Perkembangan E-Commerce terhadap Industri Jasa Transportasi dan Logistik di Era Digital," *Jurnal Pengabdian Indonesia (JPI)*, vol. 1, no. 2, pp. 352–365, Aug. 2025, doi: 10.62567/jpi.v1i2.724.

[3] G. A. Syifa and M. Tohir, "Analysis of Logistics Cost Efficiency in Indonesia's Transportation System," *Siber Journal of Transportation and Logistics (SJTL)*, vol. 3, no. 2, pp. 57–75, Sep. 2025, doi: 10.38035/sjtl.v3i2.

[4] N. L. P. D. S. Narita, I. N. Suparta, and I. N. Sukajaya, "Kekuatan Ketidakteraturan Sisi Graf Rantai C[$C_n$^(m)], n = 5, 7," Aug. 2021. doi: 10.23887/wms.v15i2.30612.

[5] A. A. D. Putra *et al.*, "Comparison of Breadth-First Search (BFS) and Depth-First Search (DFS) Algorithms for Shortest Search in Campus Maze," *JESICA: Journal of Enhanced Studies in Informatics and Computer Applications*, vol. 2, no. 2, pp. 43–51, Jul. 2025, doi: 10.47794/jesica.v2i2.14.

[6] B. K. Paju, D. Y. A. Fallo, and S. E. Mowata, "Analisis Algoritma Klasifikasi dalam Pembelajaran Sistem Informasi Geografis di Pendidikan Informatika," Kupang, Jun. 2025. doi: 10.53863/kst.v7i01.1676.

[7] D. Setiawan and T. Wijayanti Septiarini, "Systematic Literature Review: Notasi Matematika Dalam Analisis Kompleksitas Algoritma," *Prosiding Seminar Nasional Sains dan Teknologi Seri III Fakultas Sains dan Teknologi*, vol. 2, no. 1, pp. 1293–1309, Feb. 2025, Accessed: Oct. 04, 2025. [Online]. Available: https://conference.ut.ac.id/index.php/saintek/article/view/5178

[8] D. S. D. S and Asmungi, "Penentuan Rute Distribusi Pengiriman Hemodialisa Pack Untuk Menurunkan Biaya dan Waktu Pengiriman Dengan Menggunakan Metode Vehicle Routing Problems," *Jurnal Teknologi dan Manajemen Sistem Industri*, vol. 2, no. 2, pp. 113–120, Sep. 2023, doi: 10.56071/jtmsi.v2i2.614.

[9] S. P. V and S. K. Shanker P, "Degree Based Search: A Novel Graph Traversal Algorithm Using Degree Based Priority Queues," 2024. doi: 10.14569/IJACSA.2024.01507132.

[10] G. A. J. Satvika, I. N. Sukajaya, and I. G. A. Gunadi, "Improving k-nearest neighbor performance using permutation feature importance to predict student success in study," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 35, no. 3, pp. 1835–1844, Sep. 2024, doi: 10.11591/ijeecs.v35.i3.pp1835-1844.

[11] K. A. Wirakusuma and K. D. Diatmika, "Analisis Perbandingan Kompleksitas Waktu Algoritma Mst Dalam Penyusunan Jaringan Pipa Air Bersih," *Jurnal Teknologi Dan Sistem Informasi Bisnis*, vol. 7, no. 1, pp. 172–179, Jan. 2025, doi: 10.47233/jteksis.v7i1.1813.

[12] N. L. D. Sintiari, "Width Parameters on Even-Hole-Free Graphs," Jun. 2021. [Online]. Available: https://theses.hal.science/tel-03342197v1
[13] I. M. Putrama and P. Martinek, "Integrating platforms through content-based graph representation learning," *International Journal of Information Management Data Insights*, vol. 3, no. 2, Nov. 2023, doi: 10.1016/j.jjimei.2023.100200.
[14] A. V. Prolubnikov, "Graph traversals associated with iterative methods for solving systems of linear equations," Feb. 2025, doi: 10.48550.
[15] M. Pilipczuk, N. L. D. Sintiari, S. Thomassé, and N. Trotignon, "(Theta, triangle)-free and (even hole, K4)-free graphs. Part 2: Bounds on treewidth," pp. 1–20, Oct. 2020, doi: 10.1002/jgt.22675.
[16] I. K. Herdiana, I. M. Candiasa, and G. Indrawan, "Optimization of Adaptive Genetic Algorithm Parameters in Traveling Salesman Problem," *Journal of Computer Networks, Architecture and High Performance Computing*, vol. 4, no. 2, pp. 169–176, Jul. 2022, doi: 10.47709/cnahpc.v4i2.1581.
[17] D. N. S. Ariawan, G. S. Santyadiputra, and I. K. R. Arthana, "Perbandingan Performa Routing Protocol antara Mixed Routing Protocol dengan Single Routing Protocol pada Aplikasi VoIP Menggunakan Riverbed Modeler Academic Edition 17.5," *Jurnal Ilmu Komputer*, vol. 14, no. 2, pp. 11–19, 2021.
[18] R. Scheffler, "On the recognition of search trees generated by BFS and DFS," *Theor Comput Sci*, vol. 936, pp. 116–128, Nov. 2022, doi: 10.1016/j.tcs.2022.09.018.
[19] A. Mustaqim, D. B. Dinova, M. S. Fadhilah, R. A. Seivany, B. Prasetiyo, and M. A. Muslim, "Optimizing the Implementation of the BFS and DFS algorithms using the web crawler method on the kumparan site," *Journal of Soft Computing Exploration*, vol. 5, no. 2, pp. 200–206, Jul. 2024, doi: 10.52465/joscex.v5i2.309.
[20] S. Arifin, I. B. Muktyas, W. F. Al Maki, and M. K. B. M. Aziz, "Graph Coloring Program of Exam Scheduling Modeling Based on Bitwise Coloring Algorithm Using Python," *Journal of Computer Science*, vol. 18, no. 1, pp. 26–32, Feb. 2022, doi: 10.3844/jcssp.2022.26.32.
[21] M. A. P. Prasetyo, B. nebulla Syechah, N. L. D. Sintiari, and G. A. W. Wardhana, "Computing The First Zagreb Index, The Wiener Index and The Gutman Index of The Power of Dihedral Group Using Python," *Jurnal Matematika, Statistika dan Komputasi*, vol. 22, no. 1, pp. 102–113, Sep. 2025, doi: 10.20956/j.v22i1.44688.
[22] S. F. Djun, I. G. A. Gunadi, and S. Sariyasa, "Analisis Segmentasi Pelanggan pada Bisnis dengan Menggunakan Metode K-Means Clustering pada Model Data RFM," *JTIM : Jurnal Teknologi Informasi dan Multimedia*, vol. 5, no. 4, pp. 354–364, Feb. 2024, doi: 10.35746/jtim.v5i4.434.