Optimalisasi Biaya Pengisian Bahan Bakar serta Waktu dan Jarak Tempuh Rute Perjalanan: Integrasi *Dynamic Programming* dan *A* Pathfinding Algorithm*

Muhammad Raihan Nazhim Oktana - 13523021[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
E-mail: [1]m.raihannazhimoktana@gmail.com, 13523021@std.stei.itb.ac.id

Abstrak—Strategi Algoritma adalah suatu cabang materi dalam informatika yang memiliki banyak aplikasi dalam dunia teknologi. Beberapa ilmu yang dipelajari pada strategi algorima adalah tentang *dynamic programming* dan algoritma pencarian rute. Dynamic programming merupakan suatu cabang ilmu yang yang memungkinkan program untuk melakukan eksplorasi semua kemungkinan secara efisien. Algoritma pencarian rute merupakan cabang ilmu lain yang memungkinkan kita untuk mencari rute valid atau mencari rute terbaik dari suatu posisi asal ke posisi tujuan. Secara lebih rinci, terdapat 3 algoritma pencarian rute yang umum, terdiri dari Uniform Cost Search (UCS), Greedy Best-First Search (GBFS), dan A*. Saat ini, teknologi transportasi telah berkembang dengan cukup pesat, pengisian bahan bakar kendaraan menjadi hal yang penting untuk diperhatikan. Di sisi lain, perkembangan ekonomi dunia yang sedang tidak baik-baik saja membuat harga bahan bakar kendaraan melonjak tinggi, sementara daya beli masyarakat masih rendah. Integrasi *dynamic programming* dan *pathfinding algorithm* dapat menjadi solusi untuk optimalisasi biaya pengisian bahan bakar serta waktu dan jarak tempuh rute perjalanan yang dilalui dengan kendaraan. Pada kesempatan kali ini, penulis ingin mencari tahu dan menganalisis integrasi dari *dynamic programming* dan *A* pathfinding algorithm* untuk optimalisasi biaya pengisian bahan bakar serta waktu dan jarak tempuh rute perjalanan. Dari hasil percobaan yang dilakukan, ditemukan bahwa integrasi *dynamic programming* dan *A* pathfinding algorithm* pada optimalisasi biaya pengisian bahan bakar serta waktu dan jarak tempuh rute perjalanan sangat baik dan bermanfaat.

Keywords—strategi algoritma, *dynamic programming*, algoritma pencarian rute, optimalisasi biaya, dan transportasi.

I. PENDAHULUAN

Strategi algoritma merupakan pendekatan umum dalam penyelesaian masalah komputasi melalui pola berpikir tertentu, seperti brute force, greedy, divide and conquer, dynamic programming, dan lainnya. Strategi ini memungkinkan perancangan solusi yang efisien terhadap berbagai persoalan kompleks secara algoritmik. Dalam implementasinya, pemilihan strategi algoritma sangat mempengaruhi performa akhir dari sistem yang dibangun, baik dari segi akurasi hasil program, efisiensi waktu eksekusi program, maupun kompleksitas ruang eksekusi program.

Dalam konteks perencanaan rute, dua strategi yang utama adalah *Dynamic Programming (DP)* dan algoritma penelusuran rute (*pathfinding*) seperti *Uniform Cost Search (UCS)*, *Greedy Best First Search (GBFS)*, dan *A**. DP memungkinkan pemecahan submasalah yang saling tumpang tindih secara efisien untuk mengeksekusi setiap kemungkinan dengan baik, sedangkan *pathfinding* melakukan pendekatan yang lebih dinamis melalui graf berbobot yang menyimpan informasi lokasi, jarak, atau biaya antar titik. Kombinasi kedua pendekatan ini memberikan potensi besar dalam pencarian solusi optimal dari berbagai kemungkinan rute yang tersedia.

Transportasi merupakan tulang punggung kehidupan sosial dan ekonomi di Indonesia. Dengan jutaan kendaraan bermotor dan infrastruktur pembangunan jalan yang tersebar luas, permasalahan seperti kemacetan, konsumsi bahan bakar yang tinggi, dan lamanya waktu tempuh menjadi tantangan utama yang perlu diselesaikan. Sistem transportasi yang tidak efisien tidak hanya menyebabkan pemborosan waktu dan energi, tetapi juga memperburuk polusi udara serta meningkatkan beban logistik nasional. Perkembangan zaman menuntut perubahan yang lebih maju terhadap teknologi yang ada.

Dari perspektif ekonomi, efisiensi dalam pemilihan rute perjalanan berpengaruh langsung terhadap pengeluaran individu maupun operasional perusahaan. Semakin optimal rute yang dipilih, baik dalam hal jarak, waktu, maupun konsumsi bahan bakar, maka semakin besar pula penghematan yang dapat dicapai. Dalam konteks nasional, penghematan bahan bakar secara kolektif berkontribusi terhadap kestabilan energi dan pengurangan subsidi negara dan berkontribusi menjaga lingkungan. Kondisi ekonomi Indonesia yang sangat timpang dan daya beli masyarakat yang sangat rendah menuntut suatu perubahan yang positif kedepannya.

Integrasi strategi algoritma dalam permasalahan optimalisasi biaya pengisian bahan bakar serta waktu dan jarak tempuh perjalanan melalui pendekatan kombinasi *dynamic programming* dan *A* pathfinding algorithm*, dapat dirancang sistem perencanaan rute yang cerdas, efisien, dan adaptif terhadap kondisi aktual. Dengan analisis kebutuhan dan fungsional yang tepat, diiringi dengan implementasi yang baik, maka ide ini dapat menjadi alternatif solusi untuk menyelesaikan permasalahan yang ada di kehidupan.

II. DASAR TEORI

A. Strategi Algoritma

Strategi algoritma merupakan pendekatan umum dalam penyelesaian masalah komputasi melalui pola berpikir tertentu, seperti brute force, greedy, divide and conquer, dynamic programming, dan lainnya. Strategi ini memungkinkan perancangan solusi yang efisien terhadap berbagai persoalan kompleks secara algoritmik. Dalam implementasinya, pemilihan strategi algoritma sangat mempengaruhi performa akhir dari sistem yang dibangun, baik dari segi akurasi hasil program, efisiensi waktu eksekusi program, maupun kompleksitas ruang eksekusi program.[1]

Setiap strategi algoritma juga mengedepankan prinsip analisis kompleksitas untuk mengevaluasi performa dalam kasus terbaik, rata-rata, dan terburuk. Dengan memahami karakteristik masalah yang memerlukan solusi optimal, cepat, atau hemat memori, developer dapat memilih strategi algoritma yang paling sesuai. Hal ini penting agar implementasi tidak hanya benar secara logika tetapi juga efisien dalam skala yang lebih besar.[1]

B. *Dynamic Programming (DP)*

Dynamic Programming adalah teknik optimalisasi yang memecah masalah menjadi submasalah yang saling tumpang tindih serta menyimpan hasil perhitungan submasalah agar tidak dihitung ulang. Dua sifat utama DP adalah *optimal substructure* yaitu solusi masalah utama dapat dibangun dari solusi submasalah dengan optimal, dan *overlapping subproblems* yaitu banyak submasalah yang sama muncul berulang. Pendekatan DP sering dibagi menjadi top-down (memoization) dan bottom-up (tabulasi) sesuai dengan kebutuhan memori dan urutan komputasi.[8]

Penerapan algoritma DP sangat cocok untuk masalah ekonomi, pemilihan rute, dan penjadwalan. Ketika kompleksitas rekursif dapat menjadi sangat besar tanpa penyimpanan hasil submasalah, DP mampu menjadi pilihan strategi yang baik. Dengan menyusun tabel atau matriks untuk menyimpan solusi parsial, DP memastikan waktu eksekusi lebih terkendali, meski mengorbankan ruang memori. Keberhasilan algoritma DP ditentukan oleh perencanaan order pengisian tabel sehingga tidak ada data yang diakses sebelum diisi atau kasus yang terlewat.[9]

C. *Backtracking Algorithm*

Backtracking adalah teknik pencarian solusi yang mencoba langkah demi langkah dan kembali (backtrack) saat menemui jalan buntu, hingga semua kemungkinan dieksplorasi atau solusi ditemukan. Algoritma ini berguna untuk masalah kombinatorial atau sebuah fase dalam algoritma traversal. *Backtracking* akan memangkas kondisi penting untuk mencegah eksplorasi cabang yang tidak mungkin menghasilkan solusi valid.[4]

Meskipun konsepnya sederhana, *backtracking* dapat menjadi sangat tidak efisien tanpa disertai heuristik atau batasan eksplisit yang jelas. Kompleksitasnya seringkali menjadi eksponensial dalam kasus terburuk, sehingga hanya cocok untuk ukuran input yang relatif kecil atau dengan pemangkasan yang baik. Dengan kodisi dan implementasi yang tepat, *backtracking* dapat menjadi metode pemecahan masalah yang sangat baik.[5]

D. *Breadth-First Search (BFS) Algorithm*

BFS merupakan algoritma pencarian menjelajahi graf atau pohon secara tingkat demi tingkat, mulai dari simpul awal, kemudian semua tetangga, lalu tetangga dari tetangga, dan seterusnya. Algoritma ini menjamin menemukan jalur terpendek (dalam jumlah sisi) pada graf tak berbobot dan melibatkan struktur antrian untuk melacak simpul yang akan diproses. Dengan kompleksitas waktu O(|V|+|E|) dan ruang O(|V|), BFS efisien untuk graf yang relatif padat.[2]

Kelemahan utama BFS adalah konsumsi memori yang sanagt besar pada graf lebar, karena semua simpul pada level yang sama harus disimpan sekaligus dalam queue. Meski demikian, untuk banyak aplikasi seperti pencarian rute tak berbiaya atau traversal pohon, BFS tetap menjadi pilihan utama karena kesederhanaan algoritma dan jaminan minimalnya langkah.[3]

E. *Depth-First Search (DFS) Algorithm*

DFS merupakan algoritma pencarian menelusuri graf atau pohon dengan mengikuti satu cabang sedalam mungkin sebelum mundur dan mengeksplorasi cabang lainnya. DFS umumnya diimplementasikan secara rekursif atau dengan stack eksplisit. Algoritma ini menggunakan kompleksitas ruang O(h) dimana h adalah kedalaman maksimum, membuatnya lebih irit memori pada graf yang sangat lebar tetapi dangkal. DFS cocok untuk deteksi siklus, keterhubungan dan permasalahan backtracking.[2]

Kelemahan utama DFS adalah tidak dapat menjamin menemukan jalur terpendek dalam graf tak berbiaya, sehingga kurang cocok untuk pencarian rute optimal. Kinerja waktu tetap O(|V|+|E|), tetapi DFS bisa “tersesat” di cabang dalam jika tidak ada batasan atau heuristik untuk mengarahkan eksplorasi yang baik.[3]

F. *Pathfinding Algorithm*

Algoritma penelusuran rute (*pathfinding*) dirancang untuk menemukan rute optimal pada graf berbobot. Algoritma ini seringkali dipakai pada sistem navigasi dan simulasi. Metode seperti Dijkstra, UCS, GBFS, dan A* menawarkan berbagai kelebihan dan kekurangan antara jaminan optimalitas dan efisiensi. Pemilihan algoritma tergantung pada kebutuhan real-time, kemampuan heuristik, dan bobot graf yang digunakan.[6]

Penerapan *pathfinding* dalam teknologi transportasi modern mengintegrasikan data secara real-time, seperti kemacetan dan penutupan jalan, untuk memperbarui bobot dan heuristik. Dengan demikian, algoritma *pathfinding* dapat menyesuaikan rute secara dinamis, memberikan estimasi waktu tiba yang lebih akurat, serta mendukung opsi multi-kriteria seperti biaya dan waktu tempuh secara sekaligus.[7]

Berikut penjelasan 3 algoritma utama *pathfinding*:

1. *Uniform Cost Search (UCS)*

   UCS adalah varian *Dijkstra's Algorithm* yang mengekspansi simpul berdasarkan biaya kumulatif terkecil. UCS akan menjamin menemukan rute optimal pada graf berbobot non-negatif. Algoritma ini tidak menggunakan heuristik, sehingga eksplorasi bisa meluas ke banyak simpul sebelum mencapai tujuan. Meski optimal, UCS cenderung akan lebih lambat daripada algoritma lain dengan heuristik.[6]
2. *Greedy Best-First Search (GBFS)*

   GBFS memilih simpul berikutnya berdasarkan nilai heuristik terkecil tanpa mempertimbangkan biaya yang telah ditempuh selama ini. Pendekatan ini sangat cepat dalam banyak kasus praktis, tetapi tidak menjamin jalur optimal karena bisa terjebak pada jalur yang tampak menjanjikan secara heuristik namun sebenarnya tidak. GBFS cocok untuk aplikasi yang menuntut respons cepat dan dapat menoleransi solusi non-optimal.[6]
3. *A* (A-Star)*

   A* menggunakan kombinasi biaya sejauh ini (g) dan estimasi sisa (h) untuk memprioritaskan eksplorasi. A* akan memberikan jaminan solusi optimal jika heuristik yang digunakan admissible dan konsisten. Algoritma ini seringkali jauh lebih efisien daripada UCS dan GBFS karena mampu memangkas cabang yang kurang menjanjikan. A* menjadi standar dalam banyak sistem navigasi dan simulasi real-time.[7]

G. Teknologi Transportasi

Teknologi transportasi modern mencakup berbagai sistem terbarukan. Sistem Intelligent Transportation Systems (ITS), telematik kendaraan, dan aplikasi ride-hailing memanfaatkan algoritma komputasi untuk optimasi rute dan biaya. Dengan integrasi GPS, sensor lalu lintas, dan data cuaca, sistem ini dapat menyesuaikan rekomendasi rute secara real-time.[10]

Selain itu, tren kendaraan otonom dan mobilitas sebagai layanan (MaaS) menuntut algoritma *pathfinding* yang *robust* dan adaptif terhadap kondisi yang sangat dinamis. Teknologi ini juga mendukung analisis prediktif untuk meminimalkan kemacetan, emisi karbon, dan konsumsi bahan bakar secara keseluruhan.[10]

H. Konsep Jarak, Waktu dan Kecepatan

Konsep jarak dihitung dari suatu titik menuju titik lainnya. Pada kasus graf, jarak biasanya diukur dalam bobot edge, dapat berupa jarak euclidean, manhattan, atau bobot waktu tempuh yang dipengaruhi oleh kondisi jalan. Waktu merupakan salah satu aspek penting dalam 3 konsep dasar ini sebagai satuan ukuran. Variabel waktu dipengaruhi oleh banyak faktor eksternal. Penggunaan metrik yang tepat sangat penting untuk akurasi model, misalnya heuristik admissible di algoritma A* atau bobot waktu untuk estimasi waktu tiba. Kombinasi jarak dan waktu membantu menentukan rute mana yang paling efisien sesuai kebutuhan pengguna.[11]

Kecepatan merupakan gabungan dari konsep jarak dan waktu. Kecepatan didefinisikan sebagai jarak tempuh dibagi dengan waktu tempuh. Kecepatan rata-rata diukur dari rata-rata kecepatan sebuah objek dalam rentang waktu tertentu setiap ruang waktu terkecil. Penggunaan data historis dan real-time untuk memodelkan bobot jarak, waktu, dan kecepatan memungkinkan sistem adaptif yang memberikan estimasi lebih akurat dan responsif.[11]

I. Kondisi Ekonomi Indonesia

Kondisi ekonomi Indonesia sangat dipengaruhi oleh efisiensi logistik dan distribusi, yang berkaitan langsung dengan biaya transportasi dan harga bahan bakar. Fluktuasi harga BBM akibat inflasi, ketersediaan infrastruktur jalan, dan berbagai kebijakan pemerintah dapat mempengaruhi biaya operasional kendaraan, sehingga optimalisasi biaya menjadi aspek penting untuk menekan tingkat inflasi harga barang. Implementasi algoritma optimalisasisi dalam sektor transportasi dapat menurunkan biaya distribusi, meningkatkan daya saing industri, dan mendorong pertumbuhan ekonomi nasional.[12]

Selain itu, peningkatan konektivitas antarwilayah melalui jalur transportasi yang dioptimalisasi dapat membuka peluang baru di berbagai sektor perekonomian Indonesia. Dengan optimalisasi biaya pengeluaran, kemandirian ekonomi dapat meningkat, ketimpangan ekonomi akan menurun, dan daya beli masyarakat akan meningkat seiring berjalannya waktu.[12]

III. PEMBAHASAN

A. Permodelan Masalah

Permodelan masalah yang pertama adalah terkait optimasi biaya bahan bakar dengan *dynamic programming (DP)*. Graf dibuat tidak berbiaya negatif, dengan simpul sebagai kota dan sisi berlabel jarak (km). Setiap simpul yang punya SPBU menyimpan harga per liter. State DP berupa (kota , isi_tangki) menunjukkan biaya minimum untuk mencapainya, kapasitas maksimum M liter, dan konsumsi 1 L untuk N km. Transisi dilakukan dengan isi tangki +1 L di SPBU (biaya = harga), atau pindah ke tetangga jika isi tangki lebih dari kebutuhan bahan bakar perjalanan.

Permodelan masalah yang kedua adalah terkait optimasi rute dan waktu perjalanan dengan *A* pathfinding algorithm*. Graf yang digunakan sama, tapi berbobot tanpa harga. Heuristik h(v) menggunakan jarak euclidean ke tujuan yang admissible dan konsisten. State hanya berupa kota, tiap kota v punya g(v) yang berupa jarak tempuh sejauh ini, serta f(v) = g(v) + h(v). Dibuat pula open set (min-heap berdasarkan f) dan closed set untuk visited.

B. Aspek Penting Algoritma

Pada *dynamic programming (DP)*, ada 3 aspek penting yang dibahas yaitu :

1. Optimal substructure: Solusi terbaik ke (v,f) dibangun dari solusi optimal sebelumnya.
2. Overlapping subproblems: Banyak rute berbeda akan menghasilkan state sama.

3. Pembatasan kapasitas: Dimensi DP cukup besar, yaitu |V|×(M+1), sehingga memerlukan struktur data priority queue untuk efisiensi.

Pada *A* pathfinding algorithm*, ada 3 aspek penting yang dibahas yaitu :

1. Heuristik admissible: h(v) ≤ jarak sebenarnya, sehingga selalu memberikan hasil optimal.
2. Heuristik konsisten: h(u) ≤ d(u,v)+h(v), sehingga tidak perlu revisiting node yang sama.
3. Struktur open / closed: Memilih simpul dengan f terendah memprioritaskan cabang paling menjanjikan.

C. Penjelasan Algoritma

Untuk optimasi biaya bahan bakar, dilakukan pendekatan dengan algoritma *dynamic programming (DP)*. Mula-mula, algoritma akan membuat tabel dp[kota][liter] berisi biaya minimal, semua diisi ∞ kecuali dp[start][0]=0. Lalu, digunakan priority queue yang selalu memproses state(kota,liter) dengan biaya terendah. Saat memproses state, jika di kota itu ada pom bensin dan liter < M, maka akan dipertimbangkan untuk menambah satu liter dengan menambah biaya sesuai harga. Kemudian, untuk setiap jalan ke kota tetangga, dihitung berapa liter dibutuhkan. Jika bahan bakar mencukupi, pindah ke tetangga sambil mengurangi liter dan perbarui biaya bila lebih kecil. Proses ini diulang terus sampai queue kosong atau kota tujuan terjangkau dengan biaya minimum.

Untuk optimasi rute terpendek dan waktu tersingkat, dilakukan pendekatan dengan *A* pathfinding algorithm*, Mula-mula, hitung heuristik h(v) sebagai jarak lurus ke tujuan, lalu inisialisasi g[start]=0 dan f[start]=g+h, serta masukkan start ke open set. Pada setiap langkah, ambil kota dengan f terendah, pindahkan ke closed set. Jika itu adalah tujuan, maka berhenti dan rekonstruksi jalur. Jika bukan tujuan, maka untuk setiap tetangga hitung g_baru sebagai g[v] + jarak(v,u). Jika lebih kecil dari g[u] saat ini, perbarui g[u], f[u], lalu masukkan u ke open set jika belum ada. Lakukan terus menerus hingga ditemukan solusi. Dengan cara ini, A* fokus menelusuri jalur yang paling menjanjikan berdasarkan jarak sudah ditempuh dan estimasi sisa serta otomatis memberi waktu tersingkat juga.

D. Analisis Kompleksitas Algoritma

Program yang dibuat memiliki 2 algoritma utama, yaitu DP dan A*. Berikut adalah detail kompleksitas waktu dan kompleksitas ruang dari kedua algoritma tersebut:

1. *Dynamic Programming (DP)*
   - Waktu : O((|V|·M + |E|·M)·log(|V|·M))
   - Ruang : O(|V|·M)
2. *A* Pathfinding Algorithm*
   - Waktu : O(|E|·log|V|)
   - Ruang : O(|V|)

E. Hasil Pengujian dan Kelebihan Kekurangan

Program ini telah diuji dengan berbagai test case, mulai dari test case yang valid, hingga test case yang tidak ada solusinya. Berikut adalah 2 contoh hasil pengujian test case yang telah dibuat:

```
===== Jarak Terdekat (Process Time : 0.13 ms) =====
Path   : K0 -> J1@20 -> K1 -> K3
Jarak  : 70.0 km
Waktu  : 70.0 Menit (1.17 Jam)
Biaya  : Rp80,000
Rencana Isi Ulang Bahan Bakar :
        > J1@20   : 1.0L @Rp80,000/L

===== Biaya Terkecil (Process Time : 2.38 ms) =====
Path   : K0 -> J3@20 -> K2 -> J4@20 -> K3
Jarak  : 90.0 km
Waktu  : 90.0 Menit (1.50 Jam)
Biaya  : Rp15,000
Rencana Isi Ulang Bahan Bakar :
        > J3@20   : 0.5L @Rp5,000/L
        > J4@20   : 2.5L @Rp5,000/L

===== Waktu Tersingkat (Process Time : 0.07 ms) =====
Path   : K0 -> J1@20 -> K1 -> K3
Jarak  : 70.0 km
Waktu  : 70.0 Menit (1.17 Jam)
Biaya  : Rp80,000
Rencana Isi Ulang Bahan Bakar :
        > J1@20   : 1.0L @Rp80,000/L

===== Gabungan (Process Time : 1.03 ms) =====
Path   : K0 -> J3@20 -> K2 -> J4@20 -> K3
Jarak  : 90.0 km
Waktu  : 90.0 Menit (1.50 Jam)
Biaya  : Rp15,000
Rencana Isi Ulang Bahan Bakar :
        > J3@20   : 0.5L @Rp5,000/L
        > J4@20   : 2.5L @Rp5,000/L
```

Gambar 1. Hasil Pengujian TC 1
Sumber: Dokumen Penulis / Lampiran

```
===== Jarak Terdekat (Process Time : 0.05 ms) =====
IMPOSSIBLE : Tangki Tidak Cukup & SPBU Tidak Membantu

===== Biaya Terkecil (Process Time : 0.03 ms) =====
IMPOSSIBLE : Semua Rute Memerlukan Biaya Tak Terhingga

===== Waktu Tersingkat (Process Time : 0.02 ms) =====
IMPOSSIBLE : Tangki Tidak Cukup Di Rute Tercepat

===== Gabungan (Process Time : 0.01 ms) =====
IMPOSSIBLE : Tidak Ada Rute Yang Feasible Di Candidate List
```

Gambar 2. Hasil Pengujian TC 2
Sumber: Dokumen Penulis / Lampiran

Berdasarkan hasil pengujian di atas, terlihat bahwa program ini dapat berjalan dengan baik, menemukan solusi dengan cukup akurat, serta sudah menangani TC yang tidak valid juga. Program ini berhasil menjamin solusi yang optimal dengan algoritma DP untuk biaya bahan bakar dan algoritma A* dengan heuristik admissible untuk jarak, didukung struktur modular yang memudahkan pemeliharaan serta penggunaan priority queue yang efisien, parameter seperti kapasitas tangki, konsumsi, dan harga SPBU juga fleksibel untuk eksperimen skenario. Namun, dimensi DP |V|×(M+1) bisa membebani memori dan waktu eksekusi program saat skala besar, pembulatan liter ke atas kadang dapat menyebabkan “overfill” dan tidak mendukung pengisian fraksional. Di sisi lain, akan ada kasus-kasus tertentu yang memungkinkan program error dengan memunculkan sebagian solusi saja dan tidak seluruhnya karena ketidaksempurnaan program ini.

F. Implementasi Algoritma pada Teknologi Transportasi

Perkembangan zaman membuat perkembangan teknologi transportasi semakin maju. Penerapan kedua algoritma ini mampu menjadi solusi kendaraan pintar untuk dapat menentukan rute yang lebih baik secara otomatis untuk penggunanya menentukan lokasi tempat pengisian bahan bakar dan detail perjalanan lainnya.

Implementasi ini dengan penyesuaian lebih lanjut kedepannya akan mampu memberikan solusi yang lebih optimal dan lebih baik. Tidak menutup kemungkinan bahwa dalam waktu dekat, akan muncul teknologi transportasi yang lebih modern dan lebih canggih dengan segala algoritma yang telah dikembangkan dari saat ini.

G. Dampak Positif pada Perkembangan Ekonomi Indonesia

Ekonomi Indonesia saat ini sedang tidak baik-baiknya, ketimpangan sosial sangatlah besar dan daya beli masyarakat sangatlah rendah. Penggunaan algoritma program ini secara langsung pada kendaraan ataupun hanya sebagai penentuan rute manual dapat memberikan warna baru pada kehidupan.

Efisiensi biaya dan bahan bakar yang digunakan dalam perjalanan terutama jarak jauh, akan sangat membantu memulihkan ekonomi Indonesia dengan mengurangi biaya pengeluaran masyarakat sedikit demi sedikit. Tentunya diiringi dengan upaya lainnya, maka akan sangat mungkin bahwa implementasi program ini mampu memberikan dampak yang positif pada perkembangan perekonomian Indonesia kedepannya dengan meningkatkan daya beli masyarakat.

IV. IMPLEMENTASI PROGRAM

Proses implementasi program integrasi dynamic programming dan A* pathfinding algorithm dalam teknologi transportasi modern untuk optimalisasi biaya pengisian bahan bakar, serta waktu dan jarak tempuh perjalanan membutuhkan beberapa tahap implementasi. Tahapan yang akan dijelaskan di sini mengabaikan tahapan kecil seperti menginput data permasalahan, atau semacamnya. Secara umum, tahap implementasi ini meliputi proses memodelkan graf, proses pembuatan algoritma *dynamic programming (DP)*, proses pembuatan *A* pathfinding algorithm*, serta integrasi dan optimalisasi program.

Pada tahap pertama, dilakukan permodelan graf dengan mengubah data kota, jalan, dan SPBU menjadi graf tidak berarah dengan simpul merepresentasikan kota dan sisi berlabel jarak (km) serta harga bensin (per liter) di tiap SPBU.

src > graph.py > Graph > add_node

```python
class Graph :

    # ALGORITMA LOKAL
    def __init__(self) -> None :
        # DESKRIPSI LOKAL
        # Konstruktor awal untuk inisialisasi struktur graf kosong.

        # KAMUS LOKAL
        # self.adj      : dict[int, list[tuple[int, float]]]
        # self.price    : dict[int, Optional[float]]
        # self.is_city  : dict[int, bool]
        # self.label    : dict[int, str]

        # ALGORITMA LOKAL
        self.adj = defaultdict(list)
        self.price = {}
        self.is_city = {}
        self.label = {}
```

Gambar 3. Struktur Data Graf
Sumber: Dokumen Penulis / Lampiran

src > graph.py > visible_path

```python
def build_graph(A : int , roads : list[tuple[str , int , int , float]] , pumps : list[tuple[str , float , float]]) -> Graph:

    # ALGORITMA LOKAL
    graph = Graph()
    for (k) in range(A) :
        graph.add_node(k , None , True , f"K{k}")
    byroad = defaultdict(list)
    for (code , off , pr) in pumps:
        byroad[code].append((off , pr))
    nxt = A
    for (code , u , v , d) in (roads) :
        st = sorted(byroad.get(code , []))
        nodes = [u]
        for (off , pr) in (st) :
            graph.add_node(nxt , pr , False , f"{code}@{off:.0f}")
            nodes.append(nxt)
            nxt += 1
        nodes.append(v)
        offs = [0.0]
        for (off , _) in (st) :
            offs.append(off)
        offs.append(d)
        for (a , b , oa , ob) in (zip(nodes , nodes[1:] , offs , offs[1:])) :
            graph.add_edge(a , b , ob - oa)
    return graph
```

Gambar 4. Proses Permodelan Graf
Sumber: Dokumen Penulis / Lampiran

Pada tahap kedua, dilakukan pembuatan algoritma DP dengan membuat tabel biaya dp[kota][isi_tangki] dan priority queue yang mengimplementasikan transisi isi ulang bensin dan pengurangan bensin setiap kebutuhan perjalan untuk menghitung biaya paling murah ke setiap state.

```python
def solve_dp(graph : Graph , path : list[int] , rate : float , tank : float , fuel0 : float , step : float = 0.5) -> tuple[float , list[tuple[int , float
    # ALGORITMA LOKAL
    need = []
    for (u , v) in (zip(path , path[1:])) :
        need.append(math.ceil(rate * graph.weight(u , v) / step))
    qq , ff = int(round(tank / step)) , int(round(fuel0 / step))

    @lru_cache(None)
    def dp(idx : int , fuel : int) :
        # DESKRIPSI LOKAL
        # Melakukan fungsi utama dynamic programming untuk solving isi ulang bahan bakar kendaraan.

        # KAMUS LOKAL
        # ...

        # ALGORITMA LOKAL
        if (idx == len(need)) :
            return (0.0 , [])
        else :
            req , node , price = need[idx] , path[idx] , graph.price[path[idx]]
            if ((price is None) and (fuel < req)) :
                return (math.inf , None)
            else :
                best = (math.inf , None)
                if (price is None) :
                    for i in range(0 , 1) :
                        nf = fuel + i - req
                        c_next , pl_next = dp(idx + 1 , nf)
                        if (c_next != math.inf) :
                            if (price is None) :
                                c_here = 0
                            else :
                                c_here = i * step * price
                            if (c_here + c_next < best[0]) :
                                if (price) :
                                    fill = [(node , i * step , price)]
                                else :
                                    fill = []
                                best = (c_here + c_next , fill + pl_next)
```

Gambar 5. Proses Algoritma DP 1
Sumber: Dokumen Penulis / Lampiran

```python
                else :
                    for i in range(max(0 , req - fuel) , qq - fuel + 1) :
                        nf = fuel + i - req
                        c_next , pl_next = dp(idx + 1 , nf)
                        if (c_next != math.inf) :
                            if (price is None) :
                                c_here = 0
                            else :
                                c_here = i * step * price
                            if (c_here + c_next < best[0]) :
                                if (price) :
                                    fill = [(node , i * step , price)]
                                else :
                                    fill = []
                                best = (c_here + c_next , fill + pl_next)
                return best

    return dp(0 , ff)
```

Gambar 6. Proses Algoritma DP 2
Sumber: Dokumen Penulis / Lampiran

Pada tahap ketiga, dilakukan pembuatan algoritma A* dengan menghitung heuristik antarkota, inisialisasi nilai g, h, dan f, serta struktur open/closed set. Setelah itu, dijalankan loop A* untuk menemukan rute terpendek berdasarkan kombinasi jarak tempuh dan estimasi sisa.

```python
def astar(graph : Graph , start : int , goal : int , w , h) -> tuple[list[int] , float] :
    # DESKRIPSI LOKAL
    # Mencari rute optimal dari start ke goal menggunakan la* pathfinding algorithm.

    # KAMUS LOKAL

    # ALGORITMA LOKAL
    pq = [(h.get(start , 0) , 0 , start , None)]
    best = {start : 0.0}
    par = {}
    while (pq) :
        f , g_u , u , p = heapq.heappop(pq)
        if (u not in par) :
            par[u] = p
            if (u == goal) :
                break
            else :
                for (v , d) in graph.adj[u] :
                    ng = g_u + w(u , v , d)
                    if (ng < best.get(v , math.inf)) :
                        best[v] = ng
                        heapq.heappush(pq , (ng + h.get(v , 0) , ng , v , u))
    if (goal not in par) :
        return ([] , math.inf)
    else :
        path = []
        cur = goal
        while (cur is not None) :
            path.append(cur)
            cur = par[cur]
        return (path[::-1] , best[goal])
```

Gambar 7. Proses Algoritma A*
Sumber: Dokumen Penulis / Lampiran

```python
def run_heuristics(graph : Graph , S : int , T : int , L : float , cons : float , tank : float , fuel0 : float) -> list[tuple[str , str , float , list ,

    # 2 - Heuristic Jarak Terdekat
    t0 = time.perf_counter()
    h_d = reverse_dijkstra(graph , T)
    p , d = astar(graph , S , T , (lambda u , v , l : l) , h_d)
    if (not p) :
        add_heuristics(results , "Jarak Terdekat" , False , time.perf_counter() - t0 , message = "Tidak Ada Jalur Graf")
    else :
        c , plan = solve_dp(graph , p , cons , tank , fuel0)
        if (not (c < math.inf)) :
            add_heuristics(results , "Jarak Terdekat" , False , time.perf_counter() - t0 , p , d , d / L , c , plan , "Tangki Tidak Cukup & SPBU Tidak Men
        else :
            add_heuristics(results , "Jarak Terdekat" , True , time.perf_counter() - t0 , p , d , d / L , c , plan , "")

    # 3 - Heuristic Biaya Terkecil
    t0 = time.perf_counter()
    cand = solve_yen(graph , S , T , (lambda u , v , l : l) , 10 , h_d)
    best = (math.inf , None , None , None)
    for (q) in (cand) :
        c_tmp , pl_tmp = solve_dp(graph , q , cons , tank , fuel0)
        if (c_tmp < best[0]) :
            best = (c_tmp , q , pl_tmp , path_cost(graph , q , (lambda u , v , l : l)))
    c , p8 , pl8 , d8 = best
    if (not (c < math.inf)) :
        add_heuristics(results , "Biaya Terkecil" , False , time.perf_counter() - t0 , p8 , d8 , 0 , c , pl8 , "Semua Rute Mengerlukan Biaya Tak Terhingga"
    else :
        add_heuristics(results , "Biaya Terkecil" , True , time.perf_counter() - t0 , p8 , d8 , (d8 / L) , c , pl8 , "")
```

Gambar 8. Proses Heuristik 1
Sumber: Dokumen Penulis / Lampiran

```python
def run_heuristics(graph : Graph , S : int , T : int , L : float , cons : float , tank : float , fuel0 : float) -> list[tuple[str , str , float , list , f

    # 4 - Heuristic Waktu Tersingkat
    t0 = time.perf_counter()
    h_t = {n : d / L for (n , d) in (h_d.items())}
    pW , t_h = astar(graph , S , T , (lambda u , v , l : l / L) , h_t)
    if (not pW) :
        add_heuristics(results , "Waktu Tersingkat" , False , time.perf_counter() - t0 , message = "Tidak Ada Jalur Graf")
    else :
        dW = path_cost(graph , pW , (lambda u , v , l : l))
        cW , plW = solve_dp(graph , pW , cons , tank , fuel0)
        if (not (cW < math.inf)) :
            add_heuristics(results , "Waktu Tersingkat" , False , time.perf_counter() - t0 , pW , dW , t_h , cW , plW , "Tangki Tidak Cukup Di Rute Tercep
        else :
            add_heuristics(results , "Waktu Tersingkat" , True , time.perf_counter() - t0 , pW , dW , t_h , cW , plW , "")

    # 5 - Heuristic Gabungan
    t0 = time.perf_counter()
    metrics = []
    for q in cand :
        d = path_cost(graph , q , (lambda u , v , l : l))
        t = d / L
        c , _ = solve_dp(graph , q , cons , tank , fuel0)
        if (c < math.inf) :
            metrics.append((q , d , t , c))
    if (not metrics) :
        add_heuristics(results , "Gabungan" , False , time.perf_counter() - t0 , message = "Tidak Ada Rute Yang Feasible Di Candidate List")
    else :
        md = min(m[1] for m in metrics)
        mt = min(m[2] for m in metrics)
        mc = min(m[3] for m in metrics)
        pG , dG , tG , cG = min(metrics , key = (lambda m : safe_div(m[1] , md) + safe_div(m[2] , mt) + safe_div(m[3] , mc)))
        plG = solve_dp(graph , pG , cons , tank , fuel0)[1]
        add_heuristics(results , "Gabungan" , True , time.perf_counter() - t0 , pG , dG , tG , cG , plG)
```

Gambar 9. Proses Heuristik 2
Sumber: Dokumen Penulis / Lampiran

Pada tahap keempat, dilakukan penggabungan hasil dari algoritma DP dan A* seperti pada gambar proses heuristik 1 dan 2 untuk menilai trade-off biaya dengan waktu/tempuh, lalu melakukan pengujian skenario, validasi kasus ekstrem, dan optimasi performa, lalu digabungkan ke main program.

```python
if (__name__ == "__main__") :
    # 1 - Input
    mode = input("Pilih Metode Input [File (1) / Terminal (2)] : ").strip()
    if (mode == "1") :
        path = input("Path File : ").strip()
        lines = read_file(path)
    else :
        lines = manual_lines()
    A , B , C , S , T , L , M , N , roads , pumps = lines_to_data(lines)

    # 2 - Graph & Heuristic
    t0 = now
    g = build_graph(A, roads, pumps)
    build_time = now - t0
    cons = 1 / N
    fuel0 = M
    results = run_heuristics(g , S , T , L , cons , M , fuel0)

    # 4 - Input Summary

    # 5 - Result
    for (name , status , t_srch , path , dist , waktu , cost , plan) in (results) :
        print(f"===== {name} (Process Time : {ms(t_srch)}) =====")
        if (status != "OK") :
            print(status)
            print()
        else :
            path_str = (" -> ".join(g.label[n] for n in visible_path(path , plan , g)))
            print(f"Path   : {path_str}")
            print(f"Jarak  : {dist:.1f} km")
            print(f"Waktu  : {mj(waktu)}")
            print(f"Biaya  : Rp{cost:,.0f}")
            if (plan) :
                print("Rencana Isi Ulang Bahan Bakar :")
                for (n , liters , harga) in (plan) :
                    print(f"\t> {g.label[n]:<7} : {liters:.1f}L @Rp{harga:,.0f}/L")
            else :
                print("Rencana Isi Ulang Bahan Bakar : (-)")
            print()
```

Gambar 10. Proses Integrasi dan Optimalisasi
Sumber: Dokumen Penulis / Lampiran

Dengan seluruh implementasi tersebut, program ini dapat berjalan dengan baik sesuai harapan. Tentunya, perlu ada banyak penyesuaian tambahan yang menghubungkan antar subprogram fungsi-fungsi tersebut. Keseluruhan implementasi program lengkap dapat dilihat pada source code di lampiran.

V. KESIMPULAN

Setelah melakukan rangkaian penelitian, penulis dapat menyimpulkan pembahasan atas berbagai percobaan dan pengamatan yang telah dilakukan sebagai berikut:

1. Penerapan materi strategi algoritma untuk memberikan optimalisasi biaya pengisian bahan bakar serta waktu dan jarak tempuh rute perjalanan dengan melakukan implementasi materi *dynamic programming* (DP) dan *A* pathfinding algorithm* sangat baik.
2. *Dynamic programming* (DP*)* terbukti menjadi salah satu solusi yang cukup optimal untuk melakukan eksplorasi semua kemungkinan dan menentukan solusi terbaiknya.
3. *A* pathfinding algorithm* terbukti menjadi salah satu algoritma pencarian rute terbaik dengan segala heuristik dan perhitungan yang ada dalam kasus permasalahan ini dibandingkan algoritma lainnya.
4. Perkembangan dunia ekonomi dan transportasi di Indonesia menjadi hal yang baik dan perlu diimbangi dengan perkembangan keilmuan lainnya. Sebagai contoh, penerapan solusi algoritma ini dalam teknologi kendaraan yang lebih modern dapat menghemat biaya yang perlu dikeluarkan oleh seseorang. Dengan itu, maka ekonomi di Indonesia dapat berkembang dan daya beli masyarakat dapat meningkat.

VI. UCAPAN TERIMA KASIH

Alhamdulillah, serangkaian proses penyusunan makalah yang berjudul "Optimalisasi Biaya Pengisian Bahan Bakar serta Waktu dan Jarak Tempuh Rute Perjalanan: Integrasi

Dynamic Programming dan *A* Pathfinding Algorithm*" di mata kuliah Strategi Algoritma telah penulis selesaikan dengan baik.

Penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc., sebagai dosen dan pembimbing yang telah memberikan arahan dan dukungan dalam penyusunan makalah ini. Terima kasih juga kepada semua pihak yang telah membantu dalam proses penyelesaian laporan ini, meskipun tidak dapat penulis sebutkan satu persatu.

Penulis berharap semoga hasil penulisan makalah ini dapat memberikan manfaat bagi kita semua dan mendorong kemajuan ilmu pengetahuan di masa yang akan datang. Di sisi lain, penulis menyadari bahwa makalah ini masih jauh dari sempurna. Oleh karena itu, penulis terbuka dan menghargai segala kritik dan saran yang konstruktif. Sekian dan terima kasih.

REFERENSI

[1] Munir, Rinaldi. (2025). Pengantar Strategi Algoritma. Diakses pada 24 Juni 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/01-Pengantar-Strategi-Algoritma-(2025).pdf

[2] Munir, Rinaldi. (2025). Breadth/Depth First Search (BFS/DFS) Bagian 1. Diakses pada 24 Juni 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/13-BFS-DFS-(2025)-Bagian1.pdf

[3] Munir, Rinaldi. (2025). Breadth/Depth First Search (BFS/DFS) Bagian 2. Diakses pada 24 Juni 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/14-BFS-DFS-(2025)-Bagian2.pdf

[4] Munir, Rinaldi. (2025). Algoritma Runut-balik (Backtracking) Bagian 1. Diakses pada 24 Juni 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/15-Algoritma-backtracking-(2025)-Bagian1.pdf

[5] Munir, Rinaldi. (2025). Algoritma Runut-balik (Backtracking) Bagian 2. Diakses pada 24 Juni 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/16-Algoritma-backtracking-(2025)-Bagian2.pdf

[6] Munir, Rinaldi. (2025). Penentuan Rute (Route/Path Planning) Bagian 1. Diakses pada 24 Juni 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf

[7] Munir, Rinaldi. (2025). Penentuan Rute (Route/Path Planning) Bagian 2. Diakses pada 24 Juni 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/26-Program-Dinamis-(2025)-Bagian2.pdf

[8] Munir, Rinaldi. (2025). Program Dinamis (Dynamic Programming) Bagian 1. Diakses pada 24 Juni 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/25-Program-Dinamis-(2025)-Bagian1.pdf

[9] Munir, Rinaldi. (2025). Program Dinamis (Dynamic Programming) Bagian 2. Diakses pada 24 Juni 2025, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/26-Program-Dinamis-(2025)-Bagian2.pdf

[10] Jia, Fan. (2022). Application of Intelligent Transportation System in Modern Transportation Management. Diakses pada 24 Juni 2025, dari https://doi.org/10.1109/ICRSS65752.2024.00034

[11] Zahira, Nashwa Aisya. (2023). Mengenal Waktu, Jarak, dan Kecepatan. Diakses pada 24 Juni 2025, dari https://pgmi.ipmafa.ac.id/2023/01/mengenal-waktu-jarak-dan-kecepatan.html

[12] Rizani, Ahmad, dkk. (2023). Efek Inflasi terhadap Daya Beli Masyarakat pada Tinjauan Ekonomi Makro. Diakses pada 24 Juni 2025, dari https://www.researchgate.net/profile/Afif-Yahya/publication/377487995_Efek_Inflasi_terhadap_Daya_Beli_Masyarakat_pada_Tinjauan_Ekonomi_Makro/links/65a94226f323f74ff1c8625b/Efek-Inflasi-terhadap-Daya-Beli-Masyarakat-pada-Tinjauan-Ekonomi-Makro.pdf

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Juni 2025

Muhammad Raihan Nazhim Oktana
13523021

LAMPIRAN

[1] Source Code: https://github.com/RNXFreeze/Makalah-IF2211-StrategiAlgoritma

[2] Link Video Penjelasan: https://youtu.be/-zy_qS4EMHo