Optimasi Rute Distribusi Bahan Pokok di Kota Bandung Menggunakan Algoritma Dynamic Programming dan Brute Force

Rafa Abdussalam Danadyaksa - 13523133
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: rafaabdussalam.d@gmail.com , 13523133@std.stei.itb.ac.id

***Abstract*—Optimalisasi rute distribusi merupakan isu krusial dalam logistik, terutama untuk distribusi bahan pokok yang menuntut efisiensi waktu dan biaya. Penelitian ini mengkaji penerapan algoritma Dynamic Programming (DP) dan Brute Force untuk menyelesaikan Traveling Salesperson Problem (TSP) dalam konteks optimasi rute. Algoritma Brute Force menjamin solusi optimal dengan evaluasi semua rute, meskipun memiliki kompleksitas O(N!). Dynamic Programming menawarkan efisiensi komputasi lebih baik O(n^2 X 2^n) pada skala terbatas dengan prinsip optimal substructure. Studi ini membandingkan kinerja kedua algoritma berdasarkan waktu komputasi dan efisiensi memori. Hasil diharapkan memberikan wawasan untuk pemilihan algoritma yang tepat dalam distribusi bahan pokok.**

Keywords—Brute Force; Rute; Distribusi; Dynamic Programming.

I. Pendahuluan

Distribusi bahan pokok merupakan salah satu elemen fundamental dalam sistem rantai pasok yang bertujuan untuk memastikan ketersediaan kebutuhan dasar masyarakat, seperti pangan, sandang, dan kebutuhan rumah tangga lainnya. Efisiensi rute pengiriman menjadi faktor kunci dalam proses ini, karena dapat secara signifikan mengurangi biaya operasional, waktu tempuh, serta dampak lingkungan seperti emisi karbon yang dihasilkan dari aktivitas transportasi. Tantangan ini semakin kompleks di wilayah dengan infrastruktur yang bervariasi atau jumlah titik distribusi yang banyak, seperti daerah perkotaan padat atau wilayah terpencil dengan akses terbatas. Oleh karena itu, perencanaan rute yang optimal tidak hanya meningkatkan efisiensi logistik tetapi juga mendukung keberlanjutan ekonomi dan sosial di masyarakat.

Gambar 1 Bahan Pokok (Sumber : [1])

Makalah ini bertujuan untuk menganalisis dan membandingkan kinerja kedua algoritma dalam mengoptimasi rute distribusi bahan pokok. Perbandingan akan difokuskan pada total jarak tempuh waktu komputasi yang dibutuhkan, dan jumlah node yang dievaluasi selama proses perhitungan. Data simulasi didasarkan pada matriks jarak antar lokasi yang mewakili skenario nyata. Hasil dari makalah ini diharapkan dapat memberikan panduan praktis bagi pengelola logistik dalam menentukan rute yang optimal dalam mendistribusikan bahan pokok, metode yang sesuai berdasarkan skala operasi, serta kontribusi akademik dalam pengembangan solusi optimasi rute.

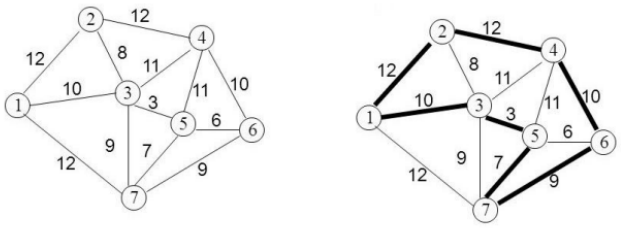
II. Dasar Teori

A. *Graf*

Graf adalah struktur matematika yang terdiri dari sekumpulan simpul (vertices) yang saling terhubung melalui sisi (edges), digunakan untuk memodelkan hubungan atau koneksi antara objek dalam berbagai aplikasi seperti jaringan transportasi, komunikasi, dan logistik. Secara formal, graf dinotasikan sebagai G = (V, E), di mana V adalah himpunan simpul dan E adalah himpunan sisi yang menghubungkan pasangan simpul.

B. *Travelling Salesperson Problem (TSP)*

Traveling Salesman Problem (TSP) merupakan permasalahan optimasi kombinatorial yang bertujuan mendapatkan rute terpendek untuk seseorang salesman guna mendatangi sekumpulan kota (simpul) serta kembali ke titik awal, dengan tiap kota/simpul yang didatangi tepat sekali. TSP dimodelkan sebagai graf berbobot lengkap G = (V, E) dengan n simpul, di mana w (u, v) menampilkan bobot (jarak ataupun pengeluaran) antara simpul u serta v. Tujuannya merupakan mendapatkan siklus Hamilton terpendek yang mencakup seluruh simpul.



Gambar 2 Graf Persoalan TSP (Sumber : [3])

C. Dynamic Programming

Dynamic Programming merupakan metode pemecahan permasalahan optimasi (maksimasi ataupun minimisasi) dengan menguraikan pemecahan sebagai serangkaian tahapan ataupun keputusan yang saling berkaitan. Kunci dari Dynamic Programming merupakan "Prinsip Optimalitas", yang menyatakan bahwa apabila solusi total merupakan optimal, hingga setiap bagian dari pemecahan tersebut juga wajib maksimal. Ini memungkinkan algoritma untuk memakai hasil maksimal dari tahap sebelumnya tanpa wajib mengulang perhitungan dari awal, sehingga menghindari perhitungan berulang untuk sub-masalah yang sama. Karakteristik permasalahan yang dapat dituntaskan dengan Dynamic Programming meliputi kemampuan untuk dipisah jadi beberapa tahap serta status, terdapatnya hubungan rekursif, serta penerapan prinsip optimalitas.

Berdasarkan pendekatan, Dynamic Programming dapat dikelompokkan menjadi dua :

1. Program Dinamis Maju (forward ataupun up- down) pendekatan ini mengawali perhitungan dari tahap awal mengarah tahap akhir, membangun solusi secara bertahap dengan memproses informasi dari awal sampai mencapai keadaan akhir.
2. Program Dinamis Mundur (backward ataupun bottom- up). Pendekatan ini bekerja dari tahap akhir mengarah tahap awal, diawali dari pemecahan akhir yang diketahui ataupun diasumsikan, kemudian melacak ke belakang guna memastikan langkah-langkah maksimal yang mengarah ke keadaan awal.

Langkah-langkah pemecahan persoalan TSP dengan Dynamic Programming :

1. Definisikan f(i, S) sebagai bobot lintasan terpendek yang berawal dari simpul i, melewati semua simpul dalam himpunan S, dan berakhir di simpul awal.
2. Kembangkan relasi secara Rekursif berdasarkan prinsip optimalitas: $f(i,S) = min\, j \in S, j \neq i\{cij + f(j, S - \{j\})\}$ di mana Cij adalah jarak dari simpul i ke j.
3. Inisiasi HashMap atau array untuk menyimpan solusi submasalah (f(i,S)) dan melacak jalur optimal
4. Cari bobot tur lengkap terkecil dengan $f(0, V - \{0\}) = mink = 1, = 1n - 1 \{c0k + f(k, V - \{0, k\})\}$ di mana V adalah himpunan semua simpul, memberikan jarak total rute yang berawal dan berakhir di simpul 0.
5. Rekonstruksi rute optimal dengan mengikuti nilai j yang meminimalkan f(i, S) pada setiap langkah

D. Brute Force

Algoritma Brute Force merupakan algoritma dengan pendekatan yang lurus ataupun straightforward guna memecahkan sesuatu permasalahan, dengan metode mengevaluasi seluruh kemungkinan pemecahan secara langsung tanpa optimasi awal. Pendekatan ini sering kali melibatkan exhaustive search, ialah prosedur pencarian menyeluruh yang mengecek setiap opsi ataupun kombinasi yang mungkin sampai penyelesaian maksimal ditemukan. Dalam konteks ini, exhaustive search memastikan seluruh rute, konfigurasi, ataupun permasalahan dieksplorasi secara sistematis, walaupun bisa menimbulkan kompleksitas waktu yang besar, semacam O(n!) pada permasalahan semacam Traveling Salesman Problem.

Langkah-langkah pemecahan persoalan TSP dengan Brute Force :

1. Tentukan titik awal
2. Hasilkan semua kemungkinan urutan kunjungan untuk n-1 kota sisanya (karena titik awal tetap). Jumlah total permutasi adalah (n-1)!, yang mencakup semua kombinasi unik.
3. Setiap permutasi di hitung total jarak rute dengan menjumlahkan jarak antar kota secara berurutan berdasarkan matriks jarak, termasuk jarak kembali ke titik awal.
4. bandingkan semua total jarak untuk memilih yang minimum, yang merupakan karakteristik utama exhaustive search karena tidak ada pemangkasan atau optimasi awal.
   Pendekatan ini secara eksostif mengecek semua rute yang mungkin (n-1)!, sesuai dengan sifat brute force yang mengevaluasi setiap kemungkinan solusi. Identifikasi jalur yang memiliki jarak terkecil sebagai solusi optimal.
5. Simpan urutan kota dalam rute terpendek beserta total jaraknya sebagai hasil akhir untuk menyelesaikan persoalan.

III. METODOLOGI

A. Batasan Penelitian

Dalam makalah ini, mempunyai beberapa batasan yang digunakan untuk menyederhanakan masalah dalam Optimasi Rute Distribusi Bahan Pokok di Kota Bandung. Batasan-batasannya sebagai berikut :

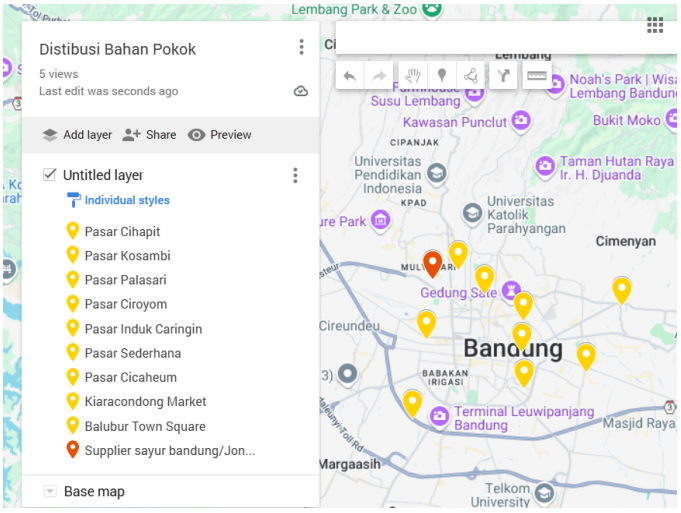
1. Algoritma yang dibahas dan diimplementasi pada makalah ini adalah Dynamic Programming dan Brute Force.

2. Dalam sekali distribusi hanya dapat mengangkut bahan untuk maksimal 10 lokasi pengiriman.
3. Makalah Ini hanya memperhitungkan Jarak Antar lokasi. Sehingga faktor lain seperti Waktu tempuh, Arus Lalu lintas tidak diperhitungkan pada makalah ini.

B. Penentuan Lokasi

Berdasarkan hasil mengidentifikasi lokasi-lokasi strategis untuk distribusi bahan pokok di Kota Bandung, penelitian ini memfokuskan pada sembilan titik utama yang mencakup berbagai pusat distribusi dan pasar tradisional sebagai titik pengiriman. Lokasi-lokasi tersebut meliputi :

1. Distributor/Supplier Sayur Bandung
2. Pasar Cihapit
3. Pasar Kosambi
4. Pasar Palasari
5. Pasar Ciroyom
6. Pasar Induk Caringin
7. Pasar Sederhana
8. Pasar Cicaheum
9. Pasar Kiaracondong
10. Balubur Town square



Gambar 3 lokasi untuk setiap cabang (Sumber : https://www.google.com/maps/d/)

C. Perancangan Solusi

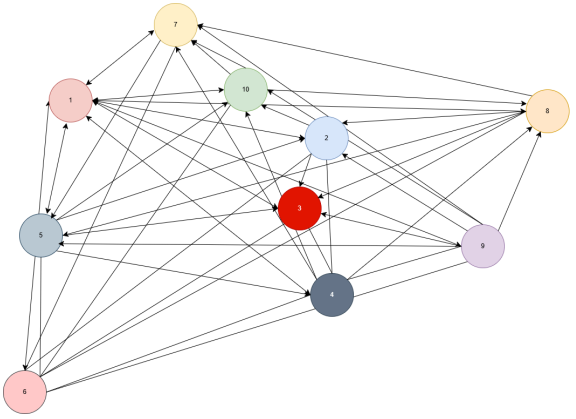
Berdasarkan data yang dikumpulkan, pengambilan data jarak dalam satuan Kilometer (KM) antara cabang dilakukan dengan memanfaatkan Google Maps sebagai alat bantu untuk menghitung rute secara realistis. Proses ini melibatkan pengukuran jarak antar lokasi cabang yang tersebar di wilayah penelitian, dengan kode cabang dari 1 hingga 10 yang dapat dilihat secara rinci pada bagian sebelumnya.

Dari/Ke	1	2	3	4	5
1	0	5.9	6	7.9	3.6
2	5.7	0	2.3	4.0	6.2
3	6.6	2.0	0	2.1	5.1
4	8.4	3.6	2	0	5.9
5	4.2	5.3	4.7	6.0	0
6	8.8	7.5	6.4	5.6	4.5
7	2.6	4.7	5.7	7.0	4.8
8	8.7	4.8	5.0	6.3	9.6
9	9.6	4.5	3.8	3.9	9.0
10	2.9	2.6	4.2	5.5	4.5

Dari/Ke	6	7	8	9	10
1	7.8	2.4	8.1	9.4	3.4
2	8.2	5.2	4.7	4.8	2.4
3	7.1	5.6	4.6	3.5	3.6
4	5.6	7.2	5.9	4.4	5.0
5	4.3	3.9	9.4	9.5	4.3
6	0	8.3	11.3	9.5	8.0
7	8.8	0	7.8	9.0	2.1
8	11.5	8.2	0	3.8	5.8
9	8.9	9.1	4.0	0	6.5
10	8.4	2.5	5.6	6.7	0

Tabel 3 Representasi Graf Bobot dengan Matriks Ketetanggan (Sumber : Dokumen Penulis)

Dari data-data yang sudah ada dibuatlah graf berarah seperti dibawah.



Gambar 4 Graf Lengkap Persoalan (Sumber : Dokumen Penulis)

D. Implementasi Algoritma Dynamic Programming

Kelas Dynamic Programming TSP mempunyai atribut statis, node Count, yang berperan buat menghitung jumlah node yang dievaluasi sepanjang rekursi, membagikan gambaran kompleksitas komputasi. Fungsi utama pada implementasi ini adalah fungsi "dynamicProgramming" yang menerima "distMatrix" (matriks jarak) sebagai input serta mengembalikan array berisi jalur terbaik, jarak total, waktu eksekusi (dalam milidetik), serta jumlah node. Tata cara ini menginisialisasi struktur informasi seperti "dp" (HashMap untuk pemecahan submasalah) serta "parent" (HashMap untuk melacak jalan maksimal) saat sebelum memanggil fungsi rekursif "tspDp".

"tspDp" merupakan inti rekursif yang menghitung jarak terpendek dari simpul disaat ini ("curr") dengan status bitmask("mask"). Tata cara ini memakai memoization untuk menyimpan hasil serta membangun jalan optimal, dan secara otomatis meningkatkan "node Count" pada setiap pemanggilan.

```java
public class DynamicProgrammingTSP {
    private static long nodeCount;

    public static Object[] dynamicProgramming(double[][] distMatrix) {
        int n = distMatrix.length;
        Map<String, Double> dp = new HashMap<>();
        Map<String, List<Integer>> parent = new HashMap<>();
        nodeCount = 0;

        long startTime = System.nanoTime();

        Object[] result = tspDp(0, 1, n, distMatrix, dp, parent);
        List<Integer> bestjalur = (List<Integer>) result[1];
        double bestDist = (Double) result[0];

        long endTime = System.nanoTime();
        double timeMs = (endTime - startTime) / 1_000_000.0;
        return new Object[]{bestjalur, bestDist, timeMs, nodeCount};
    }

    private static Object[] tspDp(int curr, int mask, int n,
    double[][] distMatrix, Map<String, Double> dp, Map<String, List<Integer>> parent) {
        nodeCount++;
        if (mask == (1 << n) - 1) {
            List<Integer> path = new ArrayList<>();
            path.add(curr);
            path.add(0); // Akhiri di 0
            return new Object[]{distMatrix[curr][0], path};
        }

        String state = curr + "," + mask;
        if (dp.containsKey(state)) {
            return new Object[]{dp.get(state), parent.get(state)};
        }

        double minDist = Double.MAX_VALUE;
        List<Integer> bestPath = new ArrayList<>();


        for (int nextCity = 0; nextCity < n; nextCity++) {
            if ((mask & (1 << nextCity)) == 0) {
                Object[] result = tspDp(nextCity, mask | (1 << nextCity), n, distMatrix, dp, parent);
                double dist = (Double) result[0];
                double totalDist = distMatrix[curr][nextCity] + dist;
                if (totalDist < minDist) {
                    minDist = totalDist;
                    bestPath = new ArrayList<>();
                    bestPath.add(curr);
                    bestPath.addAll((List<Integer>) result[1]);
                }
            }
        }

        dp.put(state, minDist);
        parent.put(state, bestPath);
        return new Object[]{minDist, bestPath};
    }
}
```

Gambar 5 Implementasi algoritma dynamic Programming (Sumber : Dokumen Penulis)

E. Implementasi Algoritma Brute Force

Di dalam kelas Brute Force TSP, terdapat atribut statis "jalurTerbaik" menyimpan informasi integer yang mewakili rute optimal, bestDistance menyimpan jarak minimum yang ditemui, serta nodeCount melacak jumlah permutasi yang dievaluasi. Fungsi utama pada implementasi algortima brute force adalah fungsi bruteForce menerima matriks jarak serta menginisialisasi variabel tersebut, setelah itu memanfaatkan generatePermutations untuk menghasilkan seluruh kemungkinan urutan kunjungan, menghitung jarak total dengan totalDistance untuk tiap rute, serta memperbarui jalurTerbaik dan bestDistance bila jarak lebih kecil ditemukan, dengan waktu eksekusi serta nodeCount dicatat untuk penilaian performa.

Selain itu, fungsi totalDistance digunakan menghitung jarak total suatu rute dengan menjumlahkan jarak antar simpul serta kembali ke titik awal, sedangkan generatePermutations secara rekursif menciptakan seluruh permutasi himpunan simpul untuk exhaustive search, menentukan seluruh opsi dievaluasi.

```java
public class BruteForceTSP {
    private static List<Integer> bestjalur;
    private static double bestDistance;
    private static long nodeCount;

    private static double totalDistance(List<Integer> jalur, double[][] distMatrix) {
        double total = 0;
        for (int i = 0; i < jalur.size() - 1; i++) {
            total += distMatrix[jalur.get(i)][jalur.get(i + 1)];
        }
        total += distMatrix[jalur.get(jalur.size() - 1)][jalur.get(0)]; // Kembali ke awal
        return total;
    }

    public static Object[] bruteForce(double[][] distMatrix) {
        int n = distMatrix.length;
        bestDistance = Double.MAX_VALUE;
        bestjalur = new ArrayList<>();
        nodeCount = 0;

        long startTime = System.nanoTime();

        List<Integer> cabang = new ArrayList<>();
        for (int i = 1; i < n; i++) {
            cabang.add(i);
        }

        List<List<Integer>> permutations = generatePermutations(cabang);
        for (List<Integer> perm : permutations) {
            nodeCount++;
            List<Integer> jalur = new ArrayList<>();
            jalur.add(0);
            jalur.addAll(perm);
            jalur.add(0);
            double dist = totalDistance(jalur, distMatrix);
            if (dist < bestDistance) {
                bestDistance = dist;
                bestjalur = new ArrayList<>(jalur);
            }
        }

        long endTime = System.nanoTime();
        double timeMs = (endTime - startTime) / 1_000_000.0; // Konversi ke milidetik
        return new Object[]{bestjalur, bestDistance, timeMs, nodeCount};
    }


    private static List<List<Integer>> generatePermutations(List<Integer> cabang) {
        List<List<Integer>> result = new ArrayList<>();
        if (cabang.isEmpty()) {
            result.add(new ArrayList<>());
            return result;
        }

        for (int i = 0; i < cabang.size(); i++) {
            int current = cabang.get(i);
            List<Integer> remaining = new ArrayList<>(cabang);
            remaining.remove(i);

            for (List<Integer> perm : generatePermutations(remaining)) {
                List<Integer> newPerm = new ArrayList<>();
                newPerm.add(current);
                newPerm.addAll(perm);
                result.add(newPerm);
            }
        }
        return result;
    }
}
```

Gambar 6 Implementasi algoritma Brute Force (Sumber : Dokumen Penulis)

IV. Hasil dan Pembahasan

A. Hasil

```
Rute: 0->4->5->3->2->8->7->1->9->6->0
Jarak: 35,30
Waktu: 612,996 ms
Node dicek: 362880
```

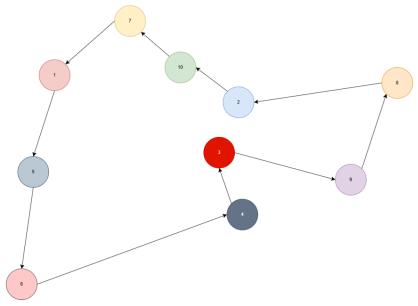
Gambar 7 Hasil kompilasi persoalan dengan algoritma Brute Force (Sumber : Dokumen penulis)

Setelah Implementasi dengan algoritma brute force didapatkan hasil rute terbaik adalah Distributor/Supplier Sayur Bandung → Pasar Ciroyom → Pasar Induk Caringin → Pasar Palasari → Pasar Kosambi → Pasar Kiaracondong → Pasar Cicaheum → Pasar Cihapit → Balubur Town square → Pasar Sederhana → Distributor/Supplier Sayur Bandung. Jarak minimum dari implementasi Dynamic Programming adalah 35.30 KM, waktu eksekusi program 612.996 ms, dan node yang dilakukan pengecekan sebanyak 362880 node.

```
Rute: 0->4->5->3->2->8->7->1->9->6->0
Jarak: 35,30
Waktu: 29,694 ms
Node dicek: 9226
```

Gambar 8 Hasil kompilasi persoalan dengan algoritma dynamic programming (Sumber : Dokumen penulis)

Setelah Implementasi dengan algoritma dynamic programming didapatkan hasil rute terbaik adalah Distributor/Supplier Sayur Bandung → Pasar Ciroyom → Pasar Induk Caringin → Pasar Palasari → Pasar Kosambi → Pasar Kiaracondong → Pasar Cicaheum → Pasar Cihapit → Balubur Town square → Pasar Sederhana → Distributor/Supplier Sayur Bandung. Jarak minimum dari implementasi Dynamic Programming adalah 35.30 KM, waktu eksekusi program 29.693 ms, dan node yang dilakukan pengecekan sebanyak 9226 node.



Gambar 9 Graf Hasil Optimasi rute dengan Program dinamis dan Brute Force (Sumber : Dokumen penulis)

B. Pembahasan

Hasil menunjukkan bahwa kedua algoritma, Brute Force dan Dynamic Programming, berhasil menemukan rute optimal yang sama 0→5→3→2→8→7→1→9→6→0 (semua node ditambah 1) dengan jarak total 35,30 Kilometer, yang mengindikasikan konsistensi solusi untuk TSP dengan 10 simpul. Perbedaan utama terletak pada efisiensi komputasi. Brute Force, dengan pendekatan exhaustive search, mengevaluasi 362.880 node dan membutuhkan waktu 612,996 ms, sesuai dengan kompleksitas waktu O(n!) yang eksponensial (untuk n = 10, 9! = 362.880). Ini menunjukkan bahwa metode ini mengecek semua permutasi rute secara menyeluruh, yang efektif untuk memastikan solusi optimal tetapi menjadi tidak praktis untuk jumlah simpul yang lebih besar.

Sebaliknya, Dynamic Programming menunjukkan performa yang jauh lebih baik dengan hanya 9.226 node dicek dan waktu eksekusi 29,694 ms, sesuai dengan kompleksitas O(n^2 X 2^n) yang lebih baik dibandingkan dengan kompleksitas dari algoritma brute force. Pendekatan ini memanfaatkan prinsip optimalitas dan memoization untuk menghindari perhitungan berulang, sehingga efisien untuk skala menengah seperti kasus ini. Perbedaan waktu eksekusi yang signifikan (sekitar 20 kali lebih cepat) menegaskan keunggulan Dynamic Programming dalam mengatasi TSP dengan jumlah simpul yang wajar.

V. Kesimpulan

Berdasarkan hasil percobaan yang sudah dilakukan, dapat disimpulkan bahwa kedua algoritma, Brute Force dan Dynamic Programming, efektif dalam mengoptimasi rute distribusi bahan pokok yang dimodelkan sebagai Traveling Salesman Problem (TSP) dengan 10 titik distribusi, menghasilkan rute distribusi bahan pokok di kota bandung yang optimal 0→5→3→2→8→7→1→9→6→0 atau Distributor/Supplier Sayur Bandung → Pasar Ciroyom → Pasar Induk Caringin → Pasar Palasari → Pasar Kosambi → Pasar Kiaracondong → Pasar Cicaheum → Pasar Cihapit → Balubur Town square → Pasar Sederhana → Distributor/Supplier Sayur Bandung. dengan jarak 35.30 KM.

Namun, Algoritma Dynamic Programming menunjukkan kelebihan yang signifikan dengan waktu eksekusi 29.694 ms dan 9.226 node yang diperiksa, dibandingkan dengan Brute Force yang memerlukan 612.996 ms serta 362.880 node, yang menunjukkan efisiensi yang lebih baik berkat pendekatan memoization dengan prinsip optimalitas. Meskipun algoritma Brute force menjamin hasil yang tepat melalui pencarian menyeluruh, kompleksitas waktu uang eksponensial menghambat kemampuannya untuk skala yang lebih besar, sementara Dynamic Programming lebih ideal untuk aplikasi yang praktis dengan jumlah titik distribusi yang sedang. Oleh karena itu, untuk mengoptimalkan rute distribusi bahan pokok dalam situasi dunia nyata, Dynamic Programming sangat disarankan, dengan Algoritma Brute Force sebagai alat validasi pada skala yang lebih kecil.

UCAPAN TERIMA KASIH

Puji dan syukur penulis panjatkan kepada ALLAH SWT atas kehendak-Nya penulisan makalah ini dapat diselesaikan dengan baik. Penulis ingin mengucapkan terima kasih untuk kedua orang tua penulis atas doa dan dukungannya memberikan kekuatan bagi penulis dalam melanjutkan pengerjaan makalah ini. Selanjutnya, penulis ingin menyampaikan ucapan terima kasih kepada para dosen mata kuliah Matematika Diskrit, Dr. Nur Ulfa Maulidevi, Pak Dr. ir. Rinaldi Munir, Pak Monterico Adrian, M.T. yang telah membimbing dan memberikan ilmu selama perkuliahan Mata Kuliah Strategi Algoritma. Terakhir, penulis ingin mengucapkan terima kasih kepada teman-teman yang telah memberi dukungan untuk menyelesaikan makalah ini.

REFERENSI

[1] Munir, Rinaldi (2024). Graf (Bag.1) Bahan Kuliah IF2120 Matematika Diskrit. Diakses pada 14 Juni 2025 diambil dari : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

[2] Munir, Rinaldi (2025). Program-Dinamis (Bag.1) Bahan Kuliah IF2211 Strategi Algoritma. Diakses pada 14 Juni 2025 diambil dari: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/25-Program-Dinamis-(2025)-Bagian1.pdf

[3] Munir, Rinaldi (2025). Program-Dinamis (Bag.2) Bahan Kuliah IF2211 Strategi Algoritma. Diakses pada 14 Juni 2025 diambil dari : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/26-Program-Dinamis-(2025)-Bagian2.pdf

[4] "Travelling Salesman Problem – Dynamic Programming-Tutorialspoint" Diakses pada 15 Juni 2025 diambil dari : https://www.tutorialspoint.com/data_structures_algorithms/travelling_salesman_problem_dynamic_programming.htm

[5] Munir, Rinaldi (2025). Algoritma-Brute-Force (Bag.1) Bahan Kuliah IF2211 Strategi Algoritma. Diakses pada 17 Juni 2025 diambil dari: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf

[6] Munir, Rinaldi (2025). Algoritma-Brute-Force (Bag.2) Bahan Kuliah IF2211 Strategi Algoritma. Diakses pada 17 Juni 2025 diambil dari: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/03-Algoritma-Brute-Force-(2025)-Bag2.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, 22 Juni 2025

Rafa Abdussalam Danadyaksa 13523133