

Optimasi Diskrit

 agungsetiabudi@ub.ac.id

Optimasi Diskrit

- Optimasi diskrit adalah cabang dari optimasi yang berfokus pada permasalahan di mana variabel keputusan hanya dapat mengambil nilai dari himpunan diskrit (misalnya bilangan bulat atau kombinasi tertentu).
- Berbeda dengan optimasi kontinu yang bekerja dalam domain real, optimasi diskrit sering ditemukan dalam berbagai aplikasi dunia nyata, seperti perencanaan rute, penjadwalan, dan desain jaringan.

Karakteristik Optimasi Diskrit

1. **Variabel Keputusan Diskrit:** Solusi hanya bisa berupa nilai dari himpunan terbatas (misalnya 0/1 dalam masalah knapsack atau permutasi dalam TSP).
2. **Fungsi Tujuan:** Dapat berupa fungsi linear atau non-linear yang harus diminimalkan atau dimaksimalkan.
3. **Kendala (Constraints):** Pembatasan yang harus dipenuhi oleh solusi, misalnya kapasitas, urutan, atau batasan logis.

Metode Penyelesaian

1. Pendekatan Exact (Optimal)

- **Brute Force:** Mencoba semua kemungkinan solusi (tidak efisien untuk skala besar).
- **Program Linier Bilangan Bulat (ILP/IP):** Menggunakan metode seperti *Branch and Bound* dan *Branch and Cut*.
- **Dynamic Programming:** Digunakan pada masalah seperti knapsack dan shortest path.

2. Pendekatan Heuristik (Mendekati Optimal)

- **Greedy Algorithm:** Memilih solusi terbaik secara lokal di setiap langkah (misalnya algoritma Prim dan Kruskal dalam MST).
- **Local Search:** Memulai dari solusi awal dan memperbaiki secara iteratif (misalnya *Hill Climbing*).

3. Pendekatan Metaheuristik (Eksplorasi Luas)

- **Simulated Annealing (SA):** Mencari solusi optimal dengan prinsip pendinginan metalurgi.
- **Genetic Algorithm (GA):** Menggunakan seleksi, mutasi, dan crossover untuk mencari solusi terbaik.
- **Particle Swarm Optimization (PSO):** Meniru perilaku kawanan dalam pencarian solusi.
- **Ant Colony Optimization (ACO):** Menggunakan pendekatan semut buatan dalam menemukan rute optimal.

Contoh Kasus Optimasi Diskrit

- **Travelling Salesman Problem (TSP):** Mencari rute terpendek yang mengunjungi semua kota sekali dan kembali ke asal.
- **Knapsack Problem:** Memilih kombinasi barang dengan nilai maksimum dalam kapasitas terbatas.
- **Vehicle Routing Problem (VRP):** Menentukan rute optimal bagi armada kendaraan yang mengirimkan barang ke lokasi pelanggan.

Metode Brute Force dalam Optimasi Diskrit

- Metode *brute force* adalah pendekatan pencarian ekshaustif di mana semua kemungkinan solusi diuji untuk menemukan solusi optimal.
- Meskipun metode ini menjamin menemukan solusi terbaik, ia memiliki kompleksitas yang tinggi, sehingga kurang efisien untuk masalah berskala besar.

Langkah-Langkah Brute Force

1. Identifikasi Ruang Solusi

- Tentukan semua kemungkinan solusi yang mungkin ada dalam permasalahan.

2. Evaluasi Semua Solusi

- Hitung nilai fungsi tujuan untuk setiap solusi yang memungkinkan.

3. Bandingkan dan Pilih Solusi Terbaik

- Cari solusi yang memberikan nilai optimal (maksimum atau minimum tergantung pada masalah).

Contoh: Knapsack Problem (0/1 Knapsack)

Diketahui sekumpulan barang dengan bobot dan nilai tertentu. Tujuannya adalah memilih kombinasi barang sehingga total nilai maksimal tanpa melebihi kapasitas tas. Misal **Kapasitas = 5 kg**

Barang	Bobot (kg)	Nilai (\$)
1	2	3
2	3	4
3	4	5
4	5	6

Langkah Penyelesaian Brute Force

1. Enumerasi Semua Kemungkinan Kombinasi Barang

Karena setiap barang bisa dipilih (1) atau tidak dipilih (0), total kombinasi adalah ($2^4 = 16$).

Berikut adalah semua kemungkinan kombinasi:

Kombinasi	Barang yang Diambil	Total Bobot	Total Nilai
0000	{}	0 kg	0
0001	{4}	5 kg	6
0010	{3}	4 kg	5
0011	{3,4}	9 kg	Tidak valid
0100	{2}	3 kg	4
0101	{2,4}	8 kg	Tidak valid
0110	{2,3}	7 kg	Tidak valid
0111	{2,3,4}	12 kg	Tidak valid
1000	{1}	2 kg	3

Kombinasi	Barang yang Diambil	Total Bobot	Total Nilai
1001	{1,4}	7 kg	Tidak valid
1010	{1,3}	6 kg	Tidak valid
1011	{1,3,4}	11 kg	Tidak valid
1100	{1,2}	5 kg	7
1101	{1,2,4}	10 kg	Tidak valid
1110	{1,2,3}	9 kg	Tidak valid
1111	{1,2,3,4}	14 kg	Tidak valid

2. Evaluasi Semua Solusi yang Valid

- Dari daftar di atas, hanya kombinasi yang tidak melebihi 5 kg yang valid.
- Solusi valid dengan nilai tertinggi adalah **{1,2}** dengan bobot **5 kg** dan nilai **7**.

3. Pilih Solusi Terbaik

- **Optimal Solution: Pilih barang {1,2} dengan total nilai 7.**

Kelebihan dan Kekurangan Brute Force

✓ Kelebihan:

- **Menjamin solusi optimal** karena semua kemungkinan dicoba.
- **Sederhana untuk diterapkan** pada skala kecil.

✗ Kekurangan:

- **Tidak efisien untuk skala besar**, karena kompleksitasnya eksponensial $O(2^n)$.
- **Membutuhkan banyak waktu dan komputasi** untuk jumlah variabel besar.

Penerapan Brute Force dalam Travelling Salesman Problem (TSP)

Travelling Salesman Problem (TSP) adalah masalah optimasi diskrit di mana seorang salesman harus mengunjungi setiap kota satu kali dan kembali ke kota awal dengan jarak perjalanan minimum.

Langkah-Langkah Penyelesaian dengan Brute Force

1. Identifikasi Semua Rute yang Mungkin

- Jika ada n kota, jumlah kemungkinan rute adalah $((n-1)!)$ (karena kota awal tetap).

2. Hitung Jarak untuk Setiap Rute

- Gunakan matriks jarak antar kota untuk menghitung total jarak setiap kemungkinan rute.

3. Bandingkan Semua Rute dan Pilih yang Terpendek

- Dari semua rute yang mungkin, pilih rute dengan total jarak minimum sebagai solusi optimal.

Contoh Kasus TSP (Brute Force)

Misalkan ada **4 kota** (A, B, C, D) dengan jarak antar kota ditunjukkan dalam tabel, tentukan rute dengan total jarak minimum, dimulai dari kota A:

	A	B	C	D
A	0	10	15	20
B	10	0	35	25
C	15	35	0	30
D	20	25	30	0

Langkah 1: Enumerasi Semua Rute

Karena kita memulai dari kota A, rute yang mungkin adalah permutasi dari {B, C, D}:

1. $A \rightarrow B \rightarrow C \rightarrow D \rightarrow A$

2. $A \rightarrow B \rightarrow D \rightarrow C \rightarrow A$

3. $A \rightarrow C \rightarrow B \rightarrow D \rightarrow A$

4. $A \rightarrow C \rightarrow D \rightarrow B \rightarrow A$

5. $A \rightarrow D \rightarrow B \rightarrow C \rightarrow A$

6. $A \rightarrow D \rightarrow C \rightarrow B \rightarrow A$

Langkah 2: Hitung Total Jarak untuk Setiap Rute

Berdasarkan matriks jarak:

1. **A → B → C → D → A:** $10 + 35 + 30 + 20 = 95$
2. **A → B → D → C → A:** $10 + 25 + 30 + 15 = 80$
3. **A → C → B → D → A:** $15 + 35 + 25 + 20 = 95$
4. **A → C → D → B → A:** $15 + 30 + 25 + 10 = 80$
5. **A → D → B → C → A:** $20 + 25 + 35 + 15 = 95$
6. **A → D → C → B → A:** $20 + 30 + 35 + 10 = 95$

Langkah 3: Pilih Rute dengan Jarak Terpendek

Dari hasil perhitungan, rute dengan jarak minimum adalah **$A \rightarrow B \rightarrow D \rightarrow C \rightarrow A$** dan **$A \rightarrow C \rightarrow D \rightarrow B \rightarrow A$** , keduanya memiliki jarak **80**.

Solusi optimal:

- **Rute:** $A \rightarrow B \rightarrow D \rightarrow C \rightarrow A$ atau $A \rightarrow C \rightarrow D \rightarrow B \rightarrow A$
- **Jarak minimum:** 80

Kelebihan dan Kekurangan Brute Force dalam TSP

✓ Kelebihan:

- Menjamin solusi optimal karena semua kemungkinan diperiksa.

✗ Kekurangan:

- Tidak efisien untuk jumlah kota besar.
- Kompleksitas **$O(n!)$** , sangat cepat meningkat jika **n** bertambah.

Metode Dynamic Programming untuk Travelling Salesman Problem (TSP)

- Dynamic Programming (DP) adalah teknik optimasi yang menyelesaikan masalah kompleks dengan membaginya menjadi submasalah yang lebih kecil dan menyimpan hasilnya agar tidak dihitung ulang (*memoization*).
- Untuk **Travelling Salesman Problem (TSP)**, DP dapat diterapkan menggunakan **Pendekatan Held-Karp (Bellman-Held-Karp Algorithm)** yang memiliki kompleksitas $O(n^2 * 2^n)$.

Langkah-Langkah Dynamic Programming dalam TSP

1. Definisikan Masalah

- Diberikan **n kota**, cari rute terpendek yang dimulai dari kota tertentu, mengunjungi semua kota sekali, dan kembali ke kota awal.
- Representasikan kota dengan indeks **$0, 1, 2, \dots, n-1$** .
- Gunakan **matriks jarak** ($\text{dist}[i][j]$) untuk menyatakan jarak antara kota **i** dan **j** .

2. Gunakan Representasi Bitmasking

- Karena ada **n kota**, kita bisa menggunakan **bitmask (set bit)** untuk melacak kota yang telah dikunjungi.
- Contoh: Jika **n = 4**, maka:
 - **0001** berarti hanya kota **0** yang dikunjungi.
 - **1011** berarti kota **0, 1, dan 3** telah dikunjungi.
 - **1111** berarti semua kota telah dikunjungi.

3. Definisikan Fungsi Rekursif dengan Memoization

Gunakan **dp[pos][mask]**, di mana:

- **pos** = kota terakhir yang dikunjungi.
- **mask** = bitmask yang menunjukkan kota yang telah dikunjungi.

Fungsi rekursifnya:

$$dp[pos][mask] = \min(dp[next][mask | (1 \ll next)] + dist[pos][next])$$

- Pindah ke kota **next** yang belum dikunjungi.
- Tambahkan jarak **dist[pos][next]**.
- Perbarui **mask** dengan menandai kota **next** telah dikunjungi.

4. Inisialisasi dan Basis Rekursi

- **Kondisi Awal:** Jika hanya satu kota (misalnya kota **0**) yang dikunjungi, maka **dp[0][1] = 0**.
- **Basis Rekursi:** Jika semua kota telah dikunjungi (**mask = 1111**), kembalikan jarak ke kota awal **dist[pos][0]**.

5. Hitung DP Secara Bottom-Up atau Top-Down

Gunakan **rekursi dengan memoization (Top-Down)** atau **iterasi dengan tabel DP (Bottom-Up)** untuk menghitung solusi optimal.

Kelebihan dan Kekurangan Dynamic Programming dalam TSP

✓ Kelebihan:

- Lebih cepat dari *brute force* ($O(n^2 * 2^n)$ vs $O(n!)$).
- Menghindari perhitungan ulang dengan *memoization*.

✗ Kekurangan:

- Masih **eksponensial**, sulit diterapkan untuk $n > 20$.
- Tidak seefisien metode heuristik seperti **Genetic Algorithm** atau **Ant Colony Optimization** untuk dataset besar.

Metode Greedy

- Metode *greedy* adalah pendekatan heuristik yang mengambil keputusan terbaik pada setiap langkah dengan harapan mencapai solusi optimal secara keseluruhan.
- Meskipun tidak selalu menjamin solusi optimal, metode ini jauh lebih efisien dibandingkan *brute force*.

Langkah-Langkah Umum Metode Greedy

1. Definisikan Masalah

- Identifikasi elemen-elemen keputusan dalam masalah.
- Tentukan fungsi tujuan yang akan dioptimalkan (maksimum/minimum).

2. Bangun Solusi Secara Bertahap

- Mulai dari keadaan awal yang kosong atau awal tertentu.

3. Pilih Langkah Greedy (Pilihan Terbaik Lokal)

- Pada setiap langkah, pilih opsi yang **paling optimal saat ini** tanpa mempertimbangkan konsekuensi jangka panjang.



4. Periksa Kelayakan Solusi

- Pastikan pilihan yang diambil tidak melanggar batasan (*constraints*).
- Jika solusi tidak layak, coba alternatif lain atau hentikan algoritma.

5. Ulangi Hingga Solusi Lengkap

- Teruskan langkah *greedy* sampai solusi akhir tercapai.

Karakteristik Metode Greedy

- Metode greedy bekerja dengan baik jika masalah memiliki **properti greedy-choice** dan **optimal substructure**, yaitu:
 -  **Greedy-choice property** → Solusi optimal global dapat diperoleh dengan membuat keputusan optimal lokal pada setiap langkah.
 -  **Optimal substructure** → Submasalah dalam masalah utama memiliki struktur optimal yang dapat digunakan untuk membangun solusi keseluruhan.

Contoh Penerapan Metode Greedy

1. **Travelling Salesman Problem (TSP)** → Pilih kota terdekat yang belum dikunjungi.
2. **Huffman Coding** → Bangun pohon Huffman dengan memilih dua frekuensi terendah pada setiap langkah.
3. **Kruskal's Algorithm (Minimum Spanning Tree)** → Pilih sisi dengan bobot terkecil yang tidak membentuk siklus.
4. **Dijkstra's Algorithm (Shortest Path)** → Pilih simpul dengan jarak minimum dan perbarui jarak simpul lainnya.
5. **Knapsack Problem (Fractional)** → Pilih item dengan nilai per unit bobot tertinggi terlebih dahulu.

Kelebihan dan Kekurangan Metode Greedy

✓ Kelebihan:

- **Cepat dan sederhana** dibandingkan metode lain seperti brute force.
- **Sering memberikan solusi optimal** untuk masalah dengan struktur yang sesuai.
- **Efisien dalam hal waktu dan sumber daya** untuk banyak aplikasi.

✗ Kekurangan:

- **Tidak selalu optimal** → Kadang hanya memberikan solusi *lokal optimum*, bukan *global optimum*.
- **Tidak mempertimbangkan keputusan masa depan** → Bisa menyebabkan solusi yang tidak ideal dalam beberapa kasus.

Contoh Kasus TSP dengan Metode Greedy

Gunakan contoh matriks jarak yang sama seperti sebelumnya:

	A	B	C	D
A	0	10	15	20
B	10	0	35	25
C	15	35	0	30
D	20	25	30	0

Langkah-Langkah Metode Greedy dalam TSP

1. Mulai dari Kota Awal

- Pilih kota awal sebagai titik pertama perjalanan.

2. Pilih Kota Terdekat yang Belum Dikunjungi

- Dari kota saat ini, pilih kota dengan jarak terpendek yang belum dikunjungi.

3. Ulangi Langkah 2 Sampai Semua Kota Dikunjungi

- Lanjutkan ke kota berikutnya yang terdekat sampai semua kota dikunjungi.

4. Kembali ke Kota Awal

- Setelah semua kota dikunjungi, kembali ke kota awal untuk menyelesaikan perjalanan.

Langkah 1: Mulai dari Kota A

- Kota awal: **A**

Langkah 2: Pilih Kota Terdekat dari A

- Pilihan:
 - $A \rightarrow B$ (10)
 - $A \rightarrow C$ (15)
 - $A \rightarrow D$ (20)
- **Pilih B** (karena jarak 10 adalah yang terpendek).

Langkah 3: Pilih Kota Terdekat dari B

- Kota yang tersisa: C, D
- Pilihan:
 - $B \rightarrow C$ (35)
 - $B \rightarrow D$ (25)
- **Pilih D** (karena jarak 25 lebih kecil dari 35).

Langkah 4: Pilih Kota Terdekat dari D

- Kota yang tersisa: C
- Pilihan:
 - $D \rightarrow C$ (30)
- **Pilih C** (karena satu-satunya yang tersisa).

Langkah 5: Kembali ke Kota A

- $C \rightarrow A (15)$

Hasil Rute & Total Jarak

Rute: **A → B → D → C → A**

Total Jarak: **10 + 25 + 30 + 15 = 80**

Perbandingan dengan Brute Force

Metode	Rute yang Ditemukan	Total Jarak
Brute Force	$A \rightarrow B \rightarrow D \rightarrow C \rightarrow A$	80 (optimal)
Greedy	$A \rightarrow B \rightarrow D \rightarrow C \rightarrow A$	80 (optimal)

- Dalam contoh ini, metode *greedy* kebetulan menemukan solusi optimal yang sama dengan *brute force*.
- Namun, untuk kasus lebih kompleks, metode *greedy* bisa saja tidak mendapatkan solusi terbaik.

Kelebihan dan Kekurangan Metode Greedy dalam TSP

✓ Kelebihan:

- **Lebih cepat daripada brute force** karena kompleksitasnya sekitar $O(n^2)$.
- **Mudah diimplementasikan** dan cocok untuk dataset besar.

✗ Kekurangan:

- **Tidak selalu menghasilkan solusi optimal**, terutama jika pilihan awal kurang optimal.
- **Tidak mempertimbangkan solusi global**, hanya memilih solusi terbaik secara lokal di setiap langkah.

Metode Local Search

- **Local Search** adalah metode optimasi yang mencari solusi **dengan memperbaiki solusi saat ini secara bertahap**, daripada mengeksplorasi seluruh ruang solusi seperti Brute Force.

1. Inisialisasi Solusi Awal

- Pilih solusi awal secara **random** atau dengan metode tertentu.
- Misalnya, dalam **Travelling Salesman Problem (TSP)**, solusi awal bisa berupa **urutan acak kota**.

2. Evaluasi Solusi

- Hitung **nilai fungsi objektif** dari solusi saat ini.
- Contoh: Dalam TSP, hitung total **jarak perjalanan**.

3. Pencarian Tetangga (Neighbor Search)

- Buat **solusi baru** dengan **modifikasi kecil** terhadap solusi saat ini.
- Contoh pada TSP: **Tukar dua kota dalam urutan perjalanan (swap)** atau **balik sebagian jalur (2-opt move)**.

4. Seleksi Solusi Baru

- Jika solusi baru lebih baik, maka terima perubahan.
- Jika tidak lebih baik, bisa tetap diterima dengan probabilitas tertentu (**Simulated Annealing**) atau ditolak.

5. Iterasi Sampai Kriteria Berhenti Terpenuhi

- Ulangi langkah **3 & 4** sampai:
 - Tidak ada perbaikan lebih lanjut (**konvergensi**).
 - Mencapai jumlah iterasi maksimum.
 - Waktu pencarian habis.

Contoh Local Search pada TSP (2-opt Move)

1. Mulai dengan urutan awal: **A → B → C → D → A**.
2. Evaluasi total jarak.
3. Tukar dua kota dalam rute (misal: ubah **A → C → B → D → A**).
4. Jika jarak lebih pendek, simpan perubahan.
5. Ulangi sampai tidak ada perbaikan.

Kelebihan dan kekurangan local search

- ✓ Cepat dibandingkan Brute Force
- ✓ Cocok untuk masalah besar (TSP, optimasi rute, dll)
- ✗ Bisa terjebak dalam solusi lokal (local optimum)