BAB II

LANDASAN TEORI

2.1 *Traveling Salesmen Problem* (TSP)

Travelling Salesman Problem (TSP) merupakan sebuah permasalahan optimasi yang dapat diterapkan pada berbagai kegiatan seperti routing. Masalah optimasi TSP terkenal dan telah menjadi standar untuk mencoba algoritma yang komputational. Pokok permasalahan dari TSP adalah seorang salesman harus mengunjungi sejumlah kota yang diketahui jaraknya satu dengan yang lainnya.

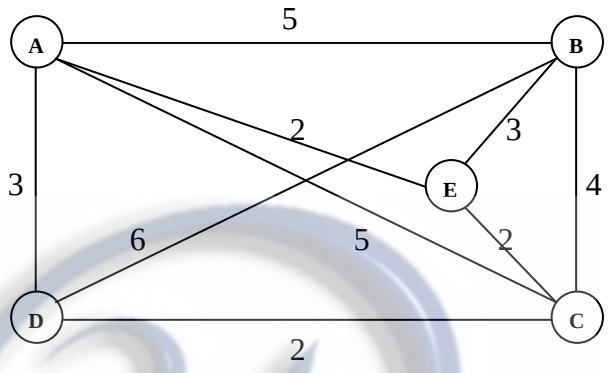
Semua kota yang ada harus dikunjungi oleh salesman tersebut dan kota tersebut hanya boleh dikunjungi tepat satu kali. Permasalahannya adalah bagaimana salesman tersebut dapat mengatur rute perjalanannya sehingga jarak yang ditempuhnya merupakan rute yang optimum yaitu jarak minimum terbaik. (Larranaga et.al 1999).

Masalah *Travelling Salesman problem* (TSP) adalah salah satu contoh yang paling banyak dipelajari dalam *combinatorial optimization.* Masalah ini mudah untuk ditanyakan tetapi sangat sulit untuk diselesaikan. TSP termasuk kelas NP-*Hard problem* dan tidak dapat diselesaikan secara optimal dalam *Polynomial computation* dengan algoritma eksak. Bila diselesaikan secara eksak waktu komputasi yang diperlukan akan meningkat secara eksponensial seiring bertambah besarnya masalah.

TSP dapat dinyatakan sebagai permasalahan dalam mencari jarak minimal sebuah tour tertutup terhadap sejumlah *n* kota dimana kota-kota yang ada hanya dikunjungi sekali. TSP di presentasikan dengan menggunakan sebuah graf lengkap dan berbobot G = (V. E) dengan V himpunan vertek yang mempresentasikan himpunan titik-titik, dan E adalah himpunan dari edge. Setiap edge (r, s) ∈ E adalah nilai (jarak) d_{rs} yang merupakan jarak dari kota r ke kota s, dengan (r, s) ∈ V. Dalam TSP simetrik (jarak dari kota *r* ke titik *s* sama dengan jarak dari titik *s* ke titik *r*), $d_{rs} = d_{rs}$ untuk semua edge (r, s) ∈ E. Misalkan

terdapat *n* buah titik maka graf tersebut memiliki $\left(\frac{n!}{((n-2)2!)}\right)$ buah egde, sesuai dengan rumus kombinasi, dan juga memiliki $\frac{(n-1)!}{2}$ buah tour yang mungkin.

Dalam sebuah graf, TSP digambarkan seperti gambar di bawah ini:

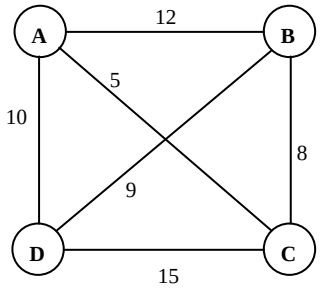


Gambar 2.1 Ilustrasi masalah TSP

Berikut adalah contoh kasus TSP: "Diberikan sejumlah kota dan jarak antar kota. Tentukan sirkuit terpendek yang harus dilalui oleh seorang pedagang itu berangkat dari sebuah kota asal dan menyinggahi setiap kota tepat satu kali dan kembali lagi ke kota asal keberangkatan."

Apabila kita mengubah contoh kasus tersebut menjadi persoalan pada graf, maka dapat dilihat bahwa kasus tersebut adalah bagaimana menentuka sirkuit Hamilton yang memiliki bobot minimum pada graf tersebut.

Seperti diketahui, bahwa untuk mencari jumlah sirkuit Hamilton di dalam graf lengkap dengan *n* vertek adalah: (n-1)!/2.



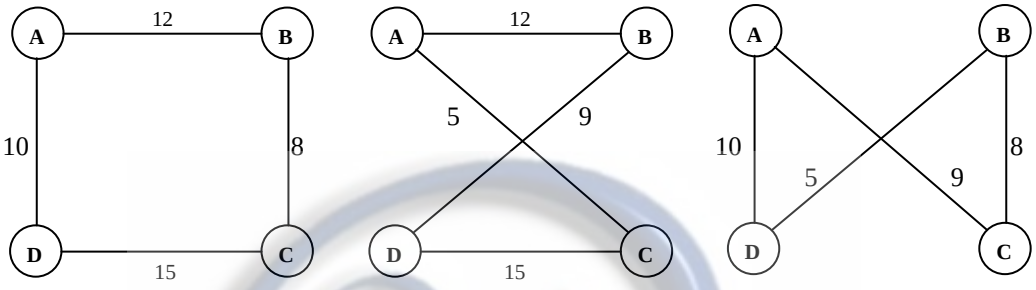
Gambar 2.2 Graf ABCD

Pada gambar diatas, graf memiliki $\frac{(4-1)!}{2} = 3$ sirkuit Hamilton, yaitu:

L1 = (*A, B, C, D, A*) atau (*A, B, C, D, A*) => panjang = 10 + 12 + 8 + 15 = 45

L2 = (*A, C, D, B, A*) atau (*A, B, D, C, A*) => panjang = 12 + 5 + 9 + 15 = 41

L3 = (*A, C, B, D, A*) atau (*A, D, B, C, A*) => panjang = 10 + 5 + 9 + 8 = 32



Gambar 2.3 Sirkuit Hamilton

Pada gambar diatas terlihat jelas bahwa sirkuit Hamilton terpendek adalah L3 = (*A, C, B, D, A*) atau (*A, D, B, C, A*) => panjang = 10 + 5 + 9 + 8 = 32. Jika jumlah vertek *n* = 20 akan terdapat (19!)/2 sirkuit Hamilton atau sekitar 6 x 10^{16} penyelesaian.

Dalam kehidupan sehari-hari, kasus TSP ini dapat diaplikasikan untuk menyelesaikan kasus lain, diantaranya yaitu:

1. Tukang POS mengambil surat di kotak pos yang tersebar pada *n* buah lokasi di berbagai sudut kota.
2. Lengan robot mengencangkan *n* buah mur pada beberapa buah peralatan mesin dalam sebuah jalur perakitan.
3. Mobil pengangkut sampah mengambil sampah pada tempat-tempat pembuangan sampah yang berada pada *n* buah lokasi di berbagai sudut kota.
4. Petugas Bank melakukan pengisian uang pada sejumlah mesin ATM di *n* buah lokasi.
5. Dan lain sebagainya.

2.1.1 Convex Hull

Convex Hull berperan penting dalam proses *Travelling Salesman Problem tour* (Vickers & Lee, 2003) dan (Rooij et al., 2003).

Convex Hull merupakan persoalan klasik dalam geometri komputasional. Definisi dari convex hull adalah poligon yang disusun dari subset titik sedemikian sehingga tidak ada titik dari himpunan awal yang berada di luar poligon tersebut (semua titik berada di batas luar atau di dalam area yang dilingkupi oleh poligon tersebut). Suatu poligon dikatakan konveks jika digambarkan garis yang menghubungkan antar titik maka tidak ada garis yang memotong garis yang menjadi batas luar poligon.

Algoritma 2.1 *Convex Hull*
1. memilih titik pertama 2. memilih titik berikutnya berdasarkan definisi : jika dibuat garis dengan titik sebelumnya maka seluruh titik lainnya tidak ada yang berada disebelah kiri jika titik tersebut sesuai maka dimasukan dalam daftar titik luar

Algoritma tersebut menggunakan pendekatan exhaustive (brute-force). kompleksitas algoritma tersebut mendekati $O(n^2)$. Algoritma tersebut dapat dioptimasi dengan membuat agar kumpulan titik-titik tersebut terurut secara lexicografis (urutkan dulu berdasarkan koordinat sumbu-X lalu untuk koordinat pada sumbu-X yang sama urutkan berdasarkan koordinat pada sumbu-Y). Sifat keterurutan ini kemudian dimanfaatkan sehingga pada setiap fase tiap titik hanya dikunjungi satu kali (kompleksitas linier). Adapun fase-fase yang perlu dilalui terdiri dari dua fase yaitu batas bagian atas (*upper boundary*) dan batas bagian bawah (*lower boundary*).

2.1.2 Metode Heuristik

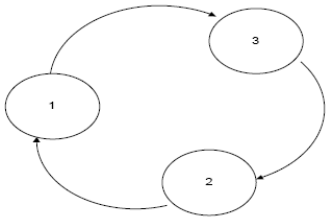
Permasalahan penentuan rute biasanya merupakan permasalahan dimana penyelesaian dengan metode *exact* seringkali akan memakan waktu yang cukup

lama untuk menyelesaikannya. Karena sebab inilah banyak para ahli yang merancang penyelesaian suatu problem dengan menggunakan merode heuristik. Metode Heuristik adalah teknik yang dirancang untuk memecahkan masalah yang mengabaikan apakah solusi dapat dibuktikan benar, tapi yang biasanya menghasilkan solusi yang baik atau memecahkan masalah yang lebih sederhana yang mengandung atau memotong dengan pemecahan masalah yang lebih kompleks. Metode Heuristik ini bertujuan untuk mendapatkan performa komputasi atau penyederhanaan konseptual, berpotensi pada biaya keakuratan atau presisi. Metode heuristik ada dua jenis yakni metode heuristik sederhana dan metaheuristik. Metode heuristik contohnya adalah *cheapest insertion, Priciest Insertion, Nearest insertion, Farthest Insertion, Nearest addition* dan *Clarke and Wright Saving Method*. Pada penelitian ini hanya menggunakan metode *cheapest insertion.*

Cheapest insertion adalah metode heuristik yang banyak digunakan untuk mencari jarak minimal antara dua titik dengan cara menyisipkan titik baru diantara kedua titik yang sudah ada sehingga terbentuk sudut atau jarak baru

Berikut ini adalah tata urutan algoritma *Cheapest Insertion* :

1. Penelusuran dimulai dari sebuah titik pertama yang dihubungkan dengan sebuah titik terakhir.
2. Dibuat sebuah hubungan *subtour* antara 2 titik tersebut. Yang dimaksud *subtour* adalah perjalanan dari titik pertama dan berakhir di titik pertama, misal (1,3) ➡ (3,2) ➡ (2,1) seperti yang tergambar dalam gambar berikut ini :



Gambar 2.4 Subtour

2. Ganti salah satu arah hubungan (*arc*) dari dua kota dengan kombinasi dua *arc*,yaitu *arc* (i,j) dengan *arc* (i,k) dan *arc* (k,j), dengan k diambil dari titik yang belum termasuk subtour dan dengan tambahan jarak terkecil. Jarak diperoleh dari penghitungan sebagai berikut :

   $C_{ik} + C_{kj} - C_{ij}$

   C_{ik} adalah jarak dari titik i ke titik k

   C_{kj} adalah jarak dari titik k ke titik j

   C_{ij} adalah jarak dari titik i ke titik k

3. Ulangi langkah 3 sampai semua titik masuk dalam subtour.

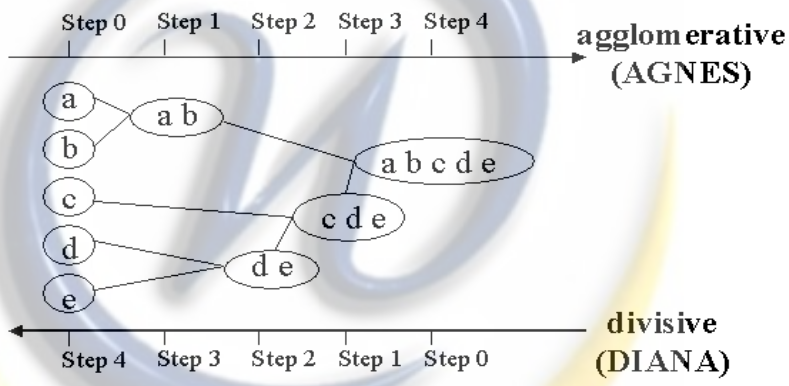
2.2 Hierarchical Clustering

Clustering dengan pendekatan hirarki adalah mengelompokkan data yang mirip dalam hirarki yang sama dan yang tidak mirip di hirarki yang agak jauh. Ada dua metode yang sering diterapkan yaitu *agglomerative hiearrarchical clustering* dan *divisive hierarchical clustering*. Agglomerative melakukan proses clustering dari *N* cluster menjadi satu kesatuan cluster, dimana *N* adalah jumlah data, sedangkan divisive melakukan proses clustering yang sebaliknya yaitu dari satu cluster menjadi *N* cluster.

Beberapa metode *hierarchical clustering* yang sering digunakan dibedakan menurut cara mereka untuk menghitung tingkat kemiripan. Ada yang menggunakan *Single Linkage, Complete Linkage, Average Linkage, Average Group Linkage* dan lain-lainnya. Seperti juga halnya dengan *partition-based clustering*, kita juga bisa memilih jenis jarak yang digunakan untuk menghitung tingkat kemiripan antar data.

Salah satu cara untuk mempermudah pengembangan dendogram untuk *hierarchical clustering* ini adalah dengan membuat *similarity matrix* yang memuat tingkat kemiripan antar data yang dikelompokkan. Tingkat kemiripan bisa dihitung dengan berbagai macam cara seperti dengan *Euclidean Distance Space*. Berangkat dari similarity matrix ini, kita bisa memilih lingkage jenis mana yang akan digunakan untuk mengelompokkan data yang dianalisa.

Teknik hirarki klastering agglomerative bekerja dengan sederetan dari penggabungan yang berurutan atau sederetan dari pembagian yang berurutan dan berawal dari objek-objek individual. Jadi pada awalnya banyaknya klaster sama dengan banyaknya objek. Objek-objek yang paling mirip dikelompokkan, dan kelompok-kelompok awal ini digabungkan sesuai dengan kemiripannya. Sewaktu kemiripan berkurang, semua subkelompok digabungkan menjadi satu klaster tunggal. Hasil-hasil dari *clustering* dapat disajikan secara grafik dalam bentuk *dendrogram* atau diagram pohon. Cabang-cabang dalam pohon menyajikan klaster dan bergabung pada node yang posisinya sepanjang sumbu jarak (similaritas) menyatakan tingkat di mana penggabungan terjadi.



Gambar 2.5 Dendrogram

Langkah-langkah dalam algoritma clustering hirarki agglomerative untuk mengelompokkan *N* objek (item/variabel):

1. Mulai dengan *N* cluster, setiap cluster mengandung entiti tunggal dan sebuah matriks simetrik dari jarak (similarities) $D = \{d_{ik}\}$ dengan tipe *NxN*.

2. Cari matriks jarak untuk pasangan cluster yang terdekat (paling mirip). Misalkan jarak antara cluster *U* dan *V* yang paling mirip adalah duv.

3. Gabungkan cluster *U* dan *V*. Label cluster yang baru dibentuk dengan *(UV).* Update entries pada matrik jarak dengan cara :

a. Hapus baris dan kolom yang bersesuaian dengan cluster *U* dan *V*

b. Tambahkan baris dan kolom yang memberikan jarak-jarak antara cluster *(UV)* dan cluster-cluster yang tersisa.

4. Ulangi langkah 2 dan 3 sebanyak *(N-1)* kali. (Semua objek akan berada dalam cluster tunggal setelah algoritma berahir). Catat identitas dari cluster yang digabungkan dan tingkat-tingkat (jarak atau similaritas) di mana penggabungan terjadi.

2.2.1 Hierarchical Clustering Average Linkage

Agerage Linkage adalah proses pengklasteran yang didasarkan pada jarak rata-rata antar objeknya. Prosedur ini hampir sama dengan *Single Linkage* maupun *Complete Linkage,* namun kriteria yang digunakan adalah rata-rata jarak seluruh individu dalam suatu klaster dengan jarak seluruh individu dalam klaster yang lain.

Average Linkage memperlakukan jarak antara dua klaster sebagai jarak rata-rata antara semua pasangan item-item dimana satu anggota dari pasangan tersebut kepunyaan tiap klaster. Mulai dengan mencari matriks jarak $D = \{d_{ik}\}$ untuk memperoleh objek-objek paling dekat (paling mirip) misalnya *U* dan *V*. Objek-objek ini digabungkan untuk membentuk klaster (*UV*). Untuk langkah dari algoritma diatas jarak-jarak antara (*UV*) dan klaster W yang lain di tentukan oleh:

$$d_{(UV)W} = \frac{\sum_i \sum_k d_{ik}}{N_{(UV)}\, N_W} \quad \text{..... (2.1)}$$

dimana d_{ik} adalah jarak antara objek *i* dalam klaster *(UV)* dan objek *k* dalam klaster *W* dan N_{uv} dan N_w berturut-turut adalah banyaknya item-item dasar klaster *(UV)* dan *W*.

2.2.2 Hierarchical Clustering Single Linkage

Input untuk algoritma *Single Linkage* bisa berujud jarak atau similarities antara pasangan-pasangan dari objek-objek. Kelompok-kelompok dibentuk dari entities individu dengan menggabungkan jarak paling pendek atau similarities (kemiripan) yang paling besar.

Pada awalnya, kita harus menemukan jarak terpendek dalam $D = \{d_{ik}\}$ dan menggabungkan objek-objek yang bersesuaian misalnya, U dan V , untuk mendapatkan *cluster* (UV). Untuk langkah (3) dari algoritma di atas jarak-jarak antara (UV) dan *cluster* W yang lain dihitung dengan cara:

$$d_{(UV)W} = min\{ d_{UW}, d_{VW} \} \qquad (2.2)$$