BAB II
LANDASAN TEORI

2.1 *Clustering Analysis*

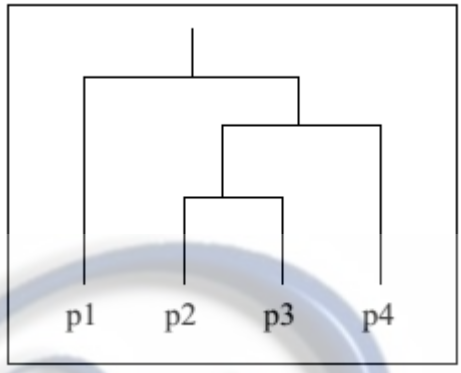
Clustering analysis merupakan metode pengelompokkan setiap objek ke dalam satu atau lebih dari satu kelompok,sehingga tiap objek yang berada dalam satu kelompok akan memiliki nilai interaksi yang sama. *Clustering analysis* bertujuan untuk membentuk kelompok dengan karakteristik yang sama. (Sharma, 1996). Pada algoritma *clustering*, data akan dikelompokkan menjadi *cluster-cluster* berdasarkan kemiripan satu data dengan data yang lain. Data yang dikelompokkan dalam satu *cluster* memiliki kemiripan yang tinggi, sedangkan antara data pada satu *cluster* dengan data pada *cluster* lainnya memiliki kemiripan yang rendah. Prinsip dari *clustering* adalah memaksimalkan kesamaan antar anggota satu kelas dan meminimumkan kesamaan antar kelas/ *cluster*. Banyak algoritma *clustering* memerlukan fungsi jarak untuk mengukur kemiripan antar data. Diperlukan juga metode untuk normalisasi bermacam - macam atribut yang dimiliki data. Fungsi jarak tersebut akan digunakan dalam proses pengerjaan program tugas akhir kali ini. Kategori algoritma *clustering* yang banyak dikenal salah satunya adalah *hierarchical clustering*.

2.1.1 *Hierarchical Clustering*

Hierarchical clustering merupakan salah satu algoritma *clustering* yang fungsinya dapat digunakan untuk meng-*cluster* dokumen (document clustering). Dari teknik *hierarchical clustering*, dapat dihasilkan suatu kumpulan partisi yang berurutan, dimana dalam kumpulan tersebut terdapat *cluster-cluster* yang mempunyai poin-poin individu, *cluster-cluster* ini berada di level yang paling bawah, Selain itu ada juga *cluster* yang didalamnya terdapat poin-poin yang dipunyai semua cluster didalamnya, *cluster* ini disebut *single cluster*, terletak di level yang paling atas.

Dalam algoritma *hierarchical clustering*, *cluster* yang berada di level yang lebih atas (*intermediate level*) dari *cluster* yang lain, dapat diperoleh dengan cara mengkombinasikan dua buah *cluster* yang berada pada level dibawahnya (Steinbach, Karypis, & Kumar, 2000). Hasil keseluruhan dari algoritma

hierarchical clustering secara grafik dapat digambarkan sebagai pohon, yang disebut dengan *dendogram*. Pohon ini secara grafik menggambarkan proses penggabungan dari *cluster – cluster* yang ada, sehingga menghasilkan *cluster* dengan level yang lebih tinggi (intermediate level). Gambar 2.1 adalah contoh gambar pohon (*dendogram*):



Gambar 2.1 *Dendogram*
(Steinbach, Karypis, & Kumar, 2000)

Ada dua metode yang sering diterapkan yaitu *agglomerative hierarchical clustering* dan *divisive hierarchical clustering*. Pada *aglomerative hierarchical*, proses hierarchical clustering dimulai dari *cluster-cluster* yang mempunyai poin-poin individu yang berada di level paling bawah. Pada setiap langkahnya, dilakukan penggabungan sebuah *cluster* dengan *cluster* lainnya, dimana *cluster - cluster* yang digabungkan berada saling berdekatan atau mempunyai tingkat/ sifat kesamaan yang paling tinggi (Steinbach, Karypis, & Kumar, 2000). Pada *divisive hierarchical clustering*, proses *hierarchical clustering* dimulai dari single *cluster* yang berada di level yang paling atas. Pada setiap langkahnya, dilakukan pemisahan (split) dari *cluster-cluster* yang ada sampai hanya tersisa *single cluster* dengan masing-masing poin individu yang dimilikinya. Dalam kasus ini, harus diputuskan, pada setiap langkahnya, *cluster* mana yang akan dipisah dan bagaimana pemisahan akan dilakukan (Steinbach, Karypis, & Kumar, 2000).

Agglomerative hierarchical clustering bekerja dengan sederetan dari penggabungan yang berurutan atau sederetan dari pembagian yang berurutan dan berawal dari objek-objek individual. Jadi pada awalnya banyaknya *cluster* sama

dengan banyaknya objek. Objek-objek yang paling mirip dikelompokkan, dan kelompok-kelompok awal ini digabungkan sesuai dengan kemiripannya. Sewaktu kemiripan berkurang, semua sub kelompok digabungkan menjadi satu *cluster* tunggal. Hasil-hasil dari *clustering* dapat disajikan secara grafik dalam bentuk *dendrogram*. Cabang-cabang dalam pohon menyajikan *cluster* dan bergabung pada node yang posisinya sepanjang sumbu jarak (similaritas) menyatakan tingkat di mana penggabungan terjadi.

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 berakhir). Catat identitas dari *cluster* yang digabungkan dan tingkat-tingkat (jarak atau similaritas) di mana penggabungan terjadi.

Beberapa metode *hierarchical clustering* yang sering digunakan dibedakan menurut cara mereka untuk menghitung tingkat kemiripan atau jarak antar kelompok. Ada yang menggunakan *ward's linkage, centroid linkage, single linkage, complete linkage, average linkage, median linkage* dan lain-lainnya.

2.1.2 *Average Linkage Clustering*

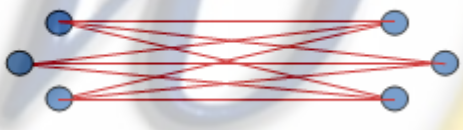
Average Linkage adalah proses pengelompokan yang didasarkan pada jarak rata-rata antar objeknya. Prosedur ini hampir sama dengan *single linkage* maupun *complete linkage,* namun kriteria yang digunakan adalah rata-rata jarak

seluruh individu dalam suatu *cluster* dengan jarak seluruh individu dalam *cluster* yang lain. (Kusrini 2007).

Average Linkage memperlakukan jarak antara dua *cluster* sebagai jarak rata-rata antara semua pasangan item-item dimana satu anggota dari pasangan tersebut kepunyaan tiap *cluster*. 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 *cluster* (*UV*). Untuk langkah dari algoritma diatas jarak-jarak antara (*UV*) dan *cluster* W yang lain di tentukan oleh:

$$d_{(UV)W} = \frac{\sum_i \sum_k d_{ik}}{N_{(uv)} N_w}$$

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

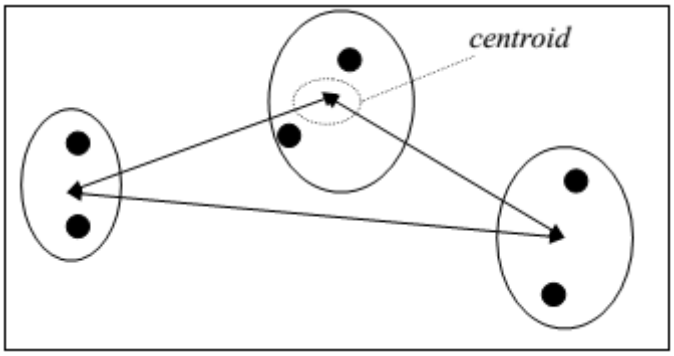


Gambar 2.2 Ilustrasi *Centroid Linkage*

(Fajar, 2012)

2.1.3 *Centroid Linkage Clustering*

Metode *centroid linkage* merupakan metode pengelompokkan berdasarkan atas nilai tengah dari data-data yang dikelompokan. Misalkan terdapat tiga kelompok seperti pada gambar 2.3, maka metode centroid akan menghitung centroid setiap kelompok dan kemudian menggabungkan kelompok dengan jarak centroid terdekat (Sharma, 1996). Pengambilan nilai tengah bukan dari jarak antara dua kelompok melainkan dari data awal yang dikelompokkan.



Gambar 2.3 Ilustrasi *Centroid Linkage*
(Sharma, 1996)

2.2 *Traveling Salesman Problem*

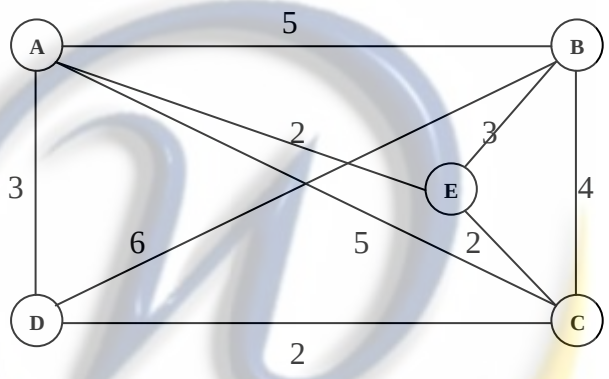
Traveling salesman problem merupakan sebuah permasalahan optimasi yang dapat diterapkan pada berbagai kegiatan seperti distribusi perdagangan. Masalah optimasi TSP terkenal dan telah menjadi standar untuk mencoba algoritma yang komputational. Pokok permasalahan dari TSP adalah seorang salesman harus 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 , 1999).

Masalah *travelling salesman problem* 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*

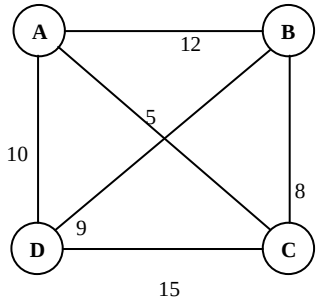
lengkap dan berbobot G = (V. E) dengan V himpunan *vertex* yang mempresentasikan himpunan titik-titik, dan E adalah himpunan dari *edge*. Setiap *edge* (r, s) ∈ E adalah nilai (jarak) d_{rs} yang merupakan jarak dari kota r ke kota s, dengan (r, s) ∈ V. Dalam TSP simetrik (jarak dari kota *r* ke titik *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 2.4 Ilustrasi masalah TSP

(Larranaga , 1999)

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* vertex adalah: (n-1)!/2. Kasus diatas akan digambarkan pada gambar 2.5.



Gambar 2.5 Graf ABCD

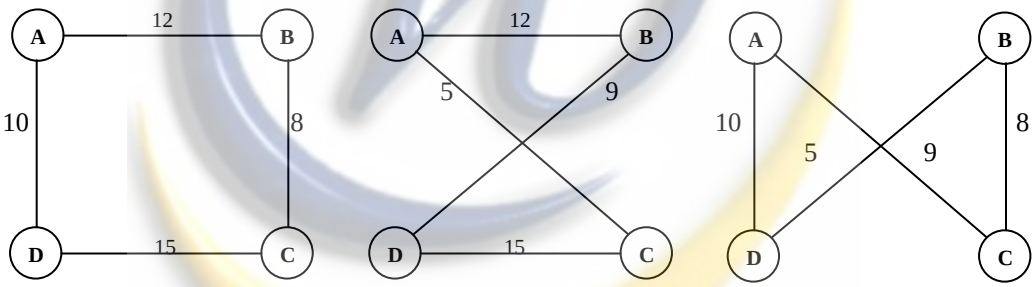
(Larranaga, 1999)

Pada gambar 2.5, 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 2.6 Sirkuit Hamilton

(Larranaga, 1999)

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 vertex *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.

2.4 Metode Heuristik

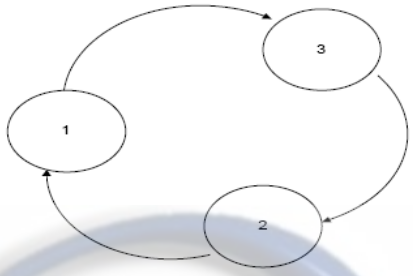
Metode Heuristik adalah teknik yang dirancang untuk memecahkan masalah yang mengabaikan apakah solusi dapat dibuktikan benar, tapi yang biasanya menghasilkan solusi yang baik atau memecahkan masalah yang lebih sederhana yang mengandung atau memotong dengan pemecahan masalah yang lebih kompleks. Metode 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*.

2.4.1 Cheapest Insertion Heuristic

Cheapest Insertion Heuristic meupakan salah satu metode untuk menyelesaikan permasalahan *Travelling Salesman Problem* (TSP). TSP sendiri merupakan masalah klasik untuk mencari rute terpendek yang biasa dilalui salesman yang ingin mengunjungi beberapa kota tanpa harus mendatangi kota yang sama lebih dari satu kali. Jika jumlah kota yang hendak didatangi hanya sedikit, permasalahan ini dapat diselesaikan dengan mudah. Tetapi akan menjadi masalah jika jumlah kota yang hendak didatangi ada 100 kota atau lebih. Oleh karena itu dengan bantuan komputer, permasalahan TSP dapat diselesaikan dengan cepat dengan mengimplementasikan algoritma *cheapest insertion heuristic*. (Kusrini, 2007). Berikut ini adalah tata urutan algoritma *cheapest insertion heuristic*:

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 dalam gambar 2.7:



Gambar 2.7 *Subtour*
(Kusrini, 2007)

3. 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.

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