BAB II
TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Tinjauan Pustaka

Kota besar mempunyai banyak masalah dalam manajemen sampah, terutama pada transportasi pengangkutan sampah. Transportasi pengangkatan sampah pada pedesaan dan perkotaan mempunyai karakter yang berbeda. Salah satu faktor yang membedakan biasanya di kota besar telah terdapat beberapa depot untuk untuk titik keberangkatan truk sampah mengambil sampah di TPS. Jumlah depot sebagai titik keberangkatan menentukan metode yang akan digunakan untuk meningkatkan efisiensi pengangkutan sampah. Pada ibu kota provinsi, seperti Kota Semarang, yang telah banyak dibangun pool di beberapa tempat untuk mengangkut sampah. Manajemen pengangkutan sampah padat di Kota Semarang, dimulai dari truk sampah keluar dari pool kemudian mengangkut sampah di TPS kemudian mengantar ke TPA.

Misalkan di sebuah kota, terdapat sejumlah TPS harus dikosongkan setiap hari dalam jangka waktu tertentu, misalkan satu jam. Masalahnya adalah menemukan jumlah minimum truk untuk melakukan ini dan waktu tersingkat untuk melakukan pengumpulan menggunakan jumlah ini truk. Sebagai contoh lain, misalkan sejumlah TPS dengan kapasitas volume tertentu, membutuhkan sejumlah truk tertentu yang harus memuat semua sampah yang ada di dalamnya. Masalahnya adalah menemukan penugasan TPS ke truk dan pengambilan jadwal tiap truk agar kapasitas tiap truk tidak terlampaui dan total jarak perjalanan diminimalkan. Beberapa variasi dari kedua masalah ini, dengan kendala waktu dan kapasitas digabungkan, adalah masalah umum di dunia nyata.

Masalah ini dapat dipecahkan sebagai TSP jika tidak ada batasan waktu dan kapasitas dan jika jumlah truk tetap (katakanlah sejumlah m). Dalam hal ini kita mendapatkan masalah m-salesmen. Namun demikian, pengelompokan (pengklasteran) dapat menerapkan metode untuk TSP untuk menemukan solusi

layak yang baik untuk masalah ini, dapat dilihat pada buku *Some simple applications of the travelling salesman problem(*Lenstra, J.K. dan Kan, A.R., 1975) (Jünger dkk,. 1995).

TSP sangat baik untuk menyelesaikan masalah perutean kendaraan (Jünger dkk,. 1995). Pada kasus pengangkutan sampah, penyelesaian masalah adalah jarak terpendek dengan aturan setiap TPS harus dikunjungi sekali dan truk harus kembali lagi ke pool. Algoritma pencarian rute terpendek selain TSP, misalkan algoritma dijkstra dan algoritma bellman ford, tidak cocok pada kasus pengangkutan sampah. Karena algoritma dijkstra dan algoritma bellman ford adalah pencarian rute terpendek dengan melewati satu atau beberapa *node*, tanpa ada aturan harus mengunjungi setiap *node*. Algoritma lain misalnya algoritma *ant colony optimization* juga tidak cocok. Karena algoritma *ant colony optimization* akan mengubah jalur pada proses optimasinya, sedangkan jalur kendaraan atau jalanan tidak bisa diubah. Contoh kasus yang mirip sebenarnya adalah *a rural postman problem*, yaitu masalah pegawai pos yang harus mengambil surat dari rumah satu kemudian mengantar ke rumah yang lain. Namun *a rural postman problem* menitik beratkan urutan dan tidak dilengkapi dengan aturan pegawai pos harus kembali ke titik awal. Lagipula, TSP juga dapat menyelesaikan permasalahan urutan pada kasus pengangkutan sampah karena sederhana, yaitu semua TPS akan diambil sampahnya (*pick-up*) dan di taruh di TPA (*drop-up*).

Banyak sekali algoritma untuk mengoptimalkan penyelesaian *Travelling Salesman Problem* (TSP), dengan setiap algoritma punya kelebihan dan kekurangan masing-masing. Algoritma yang paling sederhana untuk mengoptimalkan penyelesaian *Travelling Salesman Problem* (TSP) adalah Algoritma Brute Force. Algoritma Brute Force adalah sebuah pendekatan langsung (*straight forward*) untuk memecahkan suatu masalah, yang biasanya didasarkan pada pernyataan masalah (*problem statement*) dan definisi konsep yang dilibatkan. Pada dasarnya Algoritma Brute Force adalah alur penyelesaian suatu permasalahan dengan cara berpikir yang sederhana dan tidak membutuhkan suatu pemikiran yang lama (Usman & Oktiarso, 2019).

2.2 Dasar Teori

Dasar teori adalah sebuah konsep mendasar berupa pernyataan sistematis memiliki variabel dalam penelitian. Dasar teori menjadi landasan yang kuat dalam penelitian yang akan dilakukan. Dasar teori merupakan bagian dari penelitian yang memuat teori-teori penelitian yang berasal dari studi referensi yang memiliki fungsi sebagai kerangka teori untuk menyelesaikan pekerjaan pada penelitian. Dasar teori juga sering disebut kerangka teori. Dalam penelitian ini dasar teori meliputi *Traveling Salesman Problem* (TSP), *Algoritma Brute Force* dan *K-means Clustering*.

2.2.1. *Traveling Salesman Problem* (TSP)

Traveling salesman problem (TSP) digambarkan sebagai "Jika ada daftar kota beserta jaraknya, berapakah rute terbaik (terpendek, termurah) yang dapat diambil yang mengunjungi semua kota dan kembali ke kota awal (jalur melingkar)" (Pooja dkk., 2017). *Traveling Salesman Problem* (TSP) adalah masalah optimalisasi kombinasi dengan ketentuan seorang sales bertujuan untuk mengunjungi sejumlah kota dan kembali ke tempat asalnya. Sales tersebut harus mengunjungi setiap kota sekali dengan biaya minimal. Tujuan dari metode ini adalah mencari panjang rute terpendek yang harus dilewati sales (Rokbani dkk., 2020b).

Travelling salesman problem, disingkat TSP, memiliki karakter pemodelan di banyak cabang Matematika, Ilmu Komputer, dan Penelitian. Heuristik, pemrograman linier, dan cabang dan terikat *branch and bound algorithm*, yang masih merupakan komponen utama dari pendekatan paling sukses saat ini untuk masalah optimasi kombinatorial keras (*hard combinatorial optimization problems*), pertama kali diformulasikan untuk TSP dan digunakan untuk memecahkan contoh masalah praktis pada tahun 1954 oleh Dantzig, Fulkerson dan Johnson.

TSP Termasuk dalam kategori NP-complete. Nama "NP-complete" adalah kependekan dari "*nondeterministic polynomial-time complete*". Dalam nama ini,

"*nondeterministik*" mengacu pada mesin Turing non deterministik, suatu cara untuk memformalkan ide algoritma pencarian *brute force* secara matematis. Waktu polinomial mengacu pada jumlah waktu yang dianggap "*quick*" untuk algoritma deterministik untuk memeriksa solusi tunggal, atau untuk mesin Turing non deterministik untuk melakukan seluruh pencarian. "*complete*" mengacu pada properti untuk dapat mensimulasikan segala sesuatu di kelas kompleksitas yang sama.

Sebuah skenario masalah dapat dikatakan NP-complete jika memenuhi 3 syarat. Pertama, masalah dapat dikategorikan NP-complete dengan ketentuan kebenaran setiap solusi dapat diverifikasi dengan “cepat” (cepat disini berarti, dalam waktu polinomial atau waktu yang dapat diselesaikan komputer). Kedua, algoritma pencarian brute-force dapat menemukan solusi dengan mencoba semua solusi yang mungkin. Ketiga, masalah dapat digunakan untuk mensimulasikan setiap kasus lain yang dapat kita verifikasi dengan “cepat” bahwa solusinya pasti benar.

TSP selalu mempunyai penyelesaian *Hamiltonian cycle*. *Hamiltonian cycle* diartikan sebagai sebuah lingkaran yang melewati semua titik sekali. Penyelesaian TSP adalah *Hamiltonian cycle* dengan panjang tepian sekecil mungkin (Jünger dkk., 1995).

TSP juga dinyatakan sangat baik diterapkan untuk menyelesaikan masalah perutean kendaraan. Menurut buku *Chapter 4 The Traveling Salesman Problem* karangan Michael Jünger, Michael Jünger dan Giovanni Rinaldi, tahun 1995, TSP sangat baik diterapkan pada masalah: Pengeboran *printed circuit boards* (PCB), kristalografi sinar-X, overhaul mesin turbin gas, *the order-picking problem* di pergudangan, pengkabelan komputer, penjadwalan dengan waktu proses yang bergantung pada urutan, perutean kendaraan, masking pada PCB, kontrol 'gerakan robot’.

Contoh kasus yang membutuhkan penyelesaian TSP. Misalkan State tinggal di Gary, Indiana. Dia memiliki agen asuransi di Gary, Fort Wayne, Evansville,

Terre Haute, dan South Bend. Setiap bulan Desember, Joe mengunjungi masing-masing agen asuransinya. Jarak antara masing-masing agen (dalam mil) ditunjukkan pada Gambar 2.1. Berapa urutan mengunjungi agensinya akan meminimalkan total jarak yang ditempuh?

Day	Gary	Fort Wayne	Evansville	Terre Haute	South Bend
City 1 Gary	0	132	217	164	58
City 2 Fort Wayne	132	0	290	201	79
City 3 Evansville	217	290	0	113	303
City 4 Terre Haute	164	201	113	0	196
City 5 South Bend	58	79	303	196	0

Gambar 2.1 tabel daftar jarak antar kota

Solusinya, Joe harus menentukan urutan mengunjungi lima kota yang meminimalkan total jarak bepergian. Misalnya, Joe dapat memilih untuk mengunjungi kota-kota dalam urutan 1-3–4–5–2–1. Kemudian dia akan menempuh total 217+113+196+79+132+737 mil. Untuk mengatasi masalah Joe keliling, tentukan rumus sebagai berikut:

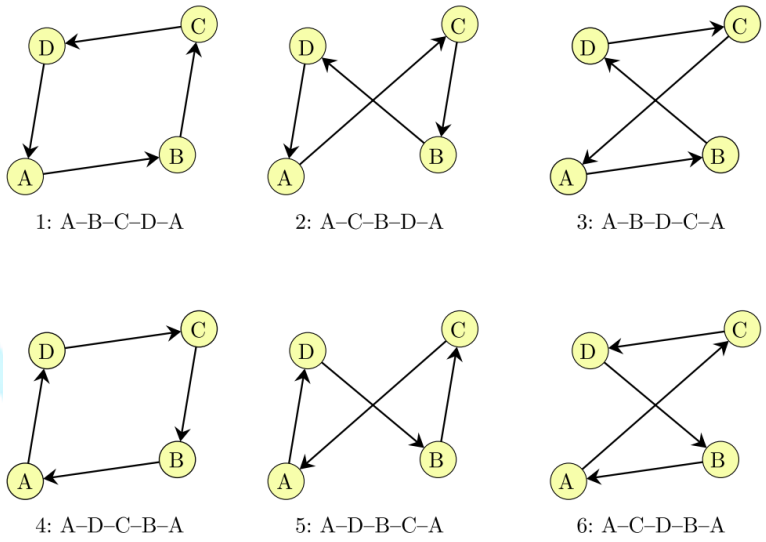
$$x_{ij} = \begin{cases} 1 & \text{if Joe leaves city } i \text{ and travels next to city } j \\ 0 & \text{otherwise} \end{cases}$$

Also, for $i \neq j$,

$$c_{ij} = \text{distance between cities } i \text{ and } j$$
$$c_{ii} = M, \text{ where } M \text{ is a large positive number}$$

Kemungkinan dapat menemukan jawaban untuk masalah Joe dengan memecahkan sebuah masalah penugasan memiliki matriks biaya yang elemen jarak dari kota asal ke kota tujuan. Kemudian Joe harus pergi dari Gary ke Fort Wayne, dari Fort Wayne ke Terre Haute, dari Terre Haute ke South Bend, dari South Bend ke Evansville, dan dari Evansville ke Gary. Solusi ini dapat ditulis sebagai 1–2–4–5–3-1. Itinerary yang dimulai dan berakhir di kota yang sama dan mengunjungi setiap kota sekali disebut rute (*tour*).

Jumlah semua kemungkinan rute bertambah kian tinggi seiring dengan bertambahnya kota yang harus dilalui sales. Jumlah semua kemungkinan adalah $(\mathrm{n}-1)!=(\mathrm{n}-1)\times(n-2)\times(n-3)\times\ldots\times 1$ rute yang berbeda untuk mengunjungi n kota dimulai dari salah satu kota (Baressi ŠEgota dkk., 2019). Gambar 2.1 memperlihatkan bahwa ada terdapat 6 kemungkinan rute yang dihasilkan untuk 4 kota yang harus dikunjungi, yaitu : A, B, C dan D. perlu dicatat bahwa 3 rute di bawah merupakan kebalikan dari rute atasnya, yang akan menghasilkan jarak yang sama. Itu artinya kita hanya perlu membandingkan setengah dari keseluruhan kemungkinan rute. Pengamatan ini menunjukan kita hanya perlu membandingkan $\frac{1}{2}(n-1)!$ rute. Lagi pula, ada varian TSP tanpa properti simetri ini. Maka pada faktanya semua kemungkinan rute harus dipertimbangkan.



Gambar 2.2 Semua rute yang mungkin terjadi untuk mengunjungi 4 kota

Untuk menemukan rute terbaik bisa menjadi tugas yang memakan waktu. Sebab TSP dan masalah serupa dapat memiliki banyak sekali kemungkinan solusi, menyelesaikannya menggunakan metode deterministik klasik seperti pencarian ekstensif dapat memakan waktu tugas. Inilah sebabnya mengapa menerapkan

metode pencarian heuristik bisa terbukti menjadi solusi yang lebih baik (Paul McMenemy dkk., 2019).

2.2.2. Algoritma Brute force

Algoritma brute force adalah proses yang mencapai solusi sempurna untuk suatu masalah dengan: menganalisis semua kemungkinan solusi kandidat. Ada kelebihan dan kekurangannya mengadopsi pendekatan semacam itu. Biasanya, pendekatan brute force mudah diterapkan, dan itu akan selalu menemukan solusi untuk masalah komputasi dengan mempertimbangkan secara iteratif semua kemungkinan solusi satu per satu. Namun, biaya komputasinya sangat bergantung pada jumlah solusi kandidat yang tersedia. Oleh karena itu, seringkali merupakan pendekatan yang relatif lambat, meskipun sederhana, untuk masalah praktis dengan ruang solusi yang luas.

Algoritma Brute force mempunyai beberapa keunggulan. Algoritma brute force dapat digunakan untuk memecahkan hampir sebagian besar masalah. Algoritma brute force juga sederhana dan mudah dimengerti. Algoritma brute force juga layak digunakan layak untuk beberapa masalah penting seperti pencarian, pengurutan, pencocokan string, perkalian matriks. Algoritma brute force menghasilkan algoritma baku (standar) untuk tugas-tugas komputasi seperti penjumlahan/perkalian N buah bilangan, menentukan elemen minimum atau maksimum di tabel.

Algoritma brute force juga memiliki kelemahan disamping keunggulannya. Walaupun sederhana dan mudah dimengerti, algoritma brute force jarang menghasilkan algoritma yang mangkus/efektif. Karena tidak efektif, seringkali algoritma brute force lambat dalam pelaksanaannya sehingga tidak dapat diterima. Algoritma brute force juga menggunakan logika dasar dan rumus sederhana, tidak sekreatif teknik pemecahan masalah lainnya

Brute Force merupakan algoritma sederhana yang melakukan pencarian menyeluruh terhadap kemungkinan yang ada (Kurniawan & Suciati, 2017). Brute

force memanfaatkan Faktorial (!) untuk menghitung banyaknya atau jumlah susunan uang dapat dibentuk dari sekumpulan atau populasi tanpa memperhatikan urutannya. Algoritma brute force menggunakan interasi mewujudkan pencarian satu nilai dari sebuah populasi besar, yang dikenal juga sebagai *linier search*. Teknik Brute Force digunakan untuk mencari penyelesaian TSP dengan membandingkan semua kemungkinan rute yang ada.

Pendekatan Brute Force, juga dikenal sebagai Pendekatan Naif, menghitung dan membandingkan semua kemungkinan permutasi rute atau jalur untuk menentukan solusi unik terpendek. Untuk menyelesaikan TSP menggunakan pendekatan Brute-Force, diharuskan menghitung jumlah total rute dan kemudian menggambar dan membuat daftar semua rute yang mungkin.

Algoritma brute force punya peran penting untuk penyelesaian TSP. Algoritma brute force membandingkan jarak seluruh rute (*searching*). Kemudian algoritma brute force juga memilih rute dengan jarak terpendek (*decisioning*). Setelah menghitung jarak setiap rute, kemudian memilih yang terpendek. Rute terpendek tersebut adalah solusi rute optimal untuk penyelesaian TSP.

2.2.3. K-means Clustering

Metode K-means merupakan metode *clustering* yang paling sederhana dan umum. Hal ini dikarenakan K-means mempunyai kemampuan mengelompokkan data dalam jumlah yang cukup besar dengan waktu komputasi yang cepat dan efisien. K-Means merupakan salah satu algoritma *clustering* dengan metode partisi (*partitioning method*) yang berbasis titik pusat (*centroid*) selain algoritma K-Medoids yang berbasis objek. Algoritma ini pertama kali diusulkan oleh MacQueen pada tahun 1967 dan dikembangkan oleh Hartigan dan Wong tahun 1975 dengan tujuan untuk dapat membagi M data point dalam N dimensi kedalam sejumlah k *cluster* dengan proses *clustering* dilakukan dengan meminimalkan jarak *sum squares* antara data dengan masing masing pusat *cluster* (*centroid-based*). Algoritma K-Means dalam penerapannya memerlukan tiga parameter yang

seluruhnya ditentukan pengguna yaitu jumlah *cluster* k, inisialisasi klaster, dan jarak sistem. Sebagian besar penggunaan K-Means dijalankan secara independen dengan inisialisasi yang berbeda menghasilkan *cluster* akhir yang berbeda karena algoritma ini secara prinsip hanya mengelompokan data menuju *local* minimal. Salah satu cara untuk mengatasi *local* minimal adalah dengan mengimplementasikan algoritma K-Means, untuk k yang diberikan, dengan beberapa nilai *initial* partisi yang berbeda dan selanjutnya dipilih partisi dengan kesalahan kuadrat terkecil (Yuan & Yang, 2019).

K-Means adalah teknik yang cukup sederhana dan cepat dalam proses *clustering* objek. Algoritma K-Means mendefinisikan *centroid* atau pusat *cluster* dari *cluster* yang menjadi rata-rata poin dari *cluster* tersebut. Dalam penerapan algoritma k-Means, jika diberikan sekumpulan data X = $\{X_1, X_2, \ldots, X_n\}$ dengan X_i = $(X_{i1}, X_{i2}, \ldots, X_{in})$ adalah sistem dalam ruang *real* R_n, maka algoritma K-Means akan menyusun partisi X dalam sejumlah k *cluster* (a priori). Setiap *cluster* memiliki titik tengah (*centroid*) yang merupakan nilai rata rata (*mean*) dari data-data dalam *cluster* tersebut. Tahapan awal, algoritma K-Means adalah memilih secara acak k buah objek sebagai *centroid* dalam data. Kemudian jarak antara objek dan *centroid* dihitung menggunakan *Euclidean distance*. Algoritma K-Means secara *iterative* meningkatkan variasi nilai dalam dalam tiap tiap *cluster* dengan objek selanjutnya ditempatkan dalam klaster yang terdekat, dihitung dari titik tengah klaster. Titik tengah baru ditentukan bila semua data telah ditempatkan dalam klaster terdekat. Proses penentuan titik tengah dan penempatan data dalam cluster diulangi sampai nilai titik tengah dari semua cluster yang terbentuk tidak berubah lagi (Kalra dkk, 2018).

Algoritma k-means:

1. Menentukan berapa banyak *cluster* k dari dataset yang akan dibagi.
2. Mengelompokkan data sehingga terbentuk K buah *cluster* dengan titik *centroid* dari setiap *cluster* merupakan titik *centroid* yang telah dipilih sebelumnya.
3. Memperbaharui nilai titik *centroid*.
4. Mengulangi langkah 2 dan 3 sampai nilai dari titik *centroid* tidak lagi berubah.