BAB II
TINJAUAN PUSTAKA

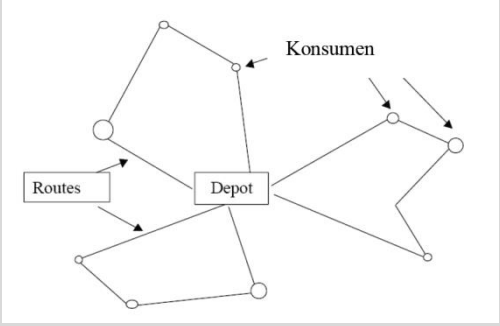
2.1 *Vehicle Routing Problem* (VRP)

2.1.1 Pengertian VRP

VRP didefinisikan sebagai permasalahan dalam penentuan rute *delivery* atau *collection* yang optimal dari depot menuju beberapa *customer* yang tersebar secara geografis dengan memperhatikan batasan operasi. Dalam aplikasinya VRP telah banyak muncul di kehidupan nyata dalam transportasi dan logistik, seperti perutean bus sekolah, perutean petugas pos, perutean truk sampah dan distribusi produk-produk kehidupan sehari-hari kepada retailer, dan sebagainya (Setiawan dan Suparno, 2009).

Tujuan dari VRP adalah menentukan sejumlah rute untuk melakukan pengiriman pada setiap konsumen, dengan mengikuti beberapa ketentuan, antara lain: (1) setiap rute berawal dan berakhir di depot, (2) setiap konsumen dikunjungi tepat satu kali oleh tepat satu kendaraan, (3) jumlah permintaan tiap rute tidak melebihi kapasitas kendaraan, dan (4) meminimumkan biaya perjalanan (Cordeau *et al*, 2002).

Gambar 2.1 merupakan contoh rute dalam VRP, dimana terdapat satu buah depot dan beberapa konsumen yang tersebar sehingga membentuk tiga buah rute yang harus dilalui oleh kendaraan.



Gambar 2.1 Contoh rute dalam VRP (Raditya,2009)

2.1.2 Formulasi model matematis VRP

Thangiah (1995) merumuskan model *mixed-integer programming* untuk permasalahan VRP. Parameter yang digunakan antara lain: K sebagai nomer kendaraan, N sebagai nomer konsumen (0 menunjukkan depot), C_i menunjukkan konsumen i, C_0 menunjukkan *depot*. Selanjutnya V_k sebagai rute kendaraan k, c_{ijk} adalah biaya *travel* antara konsumen i dan j untuk kendaraan k, q_{ik} menyatakan total *demand* kendaraan k sampai konsumen i, dan v_k adalah kapasitas maksimum kendaraan k, sedangkan variabelnya didefinisikan sebagai:

$$y_{ik} = \begin{cases} 1 & \text{Jika konsumen } i \text{ dilayani oleh kendaraan } k \\ 0 & \text{Jika tidak demikian} \end{cases}$$

$$x_{ijk} = \begin{cases} 1 & \text{Jika kendaraan } k \text{ dari konsumen } i \text{ langsung ke } j \\ 0 & \text{Jika tidak demikian} \end{cases}$$

Formulasi *mixed-integer programing* untuk VRP ini adalah:
Fungsi tujuan:

$$\min \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{k=1}^{K} c_{ij}\, x_{ijk} \tag{2.1}$$

dengan kendala – kendala:

$$\sum_{i=0}^{N} q_{ik}\, y_{ik} \leq v_k, k = 1, \dots, K \tag{2.2}$$

y_{ik} = 0 atau 1, $\quad i = 1,2, \dots, N; k = 1,2, .., K$ (2.3)

x_{ijk} = 0 atau 1, $\quad i = 1,2, \dots, N; k = 1,2, .., K$ (2.4)

$$\sum_{k=1}^{K} y_{ik} = \begin{cases} K, & i = 0 \\ 1, & i = 1, .., N \end{cases} \tag{2.5}$$

$$\sum_{i=0}^{N} x_{ijk} = y_{jk}, \quad j = 0, \dots, N; k = 1, \dots, K \tag{2.6}$$

$$\sum_{j=0}^{N} x_{ijk} = y_{ik}, \quad i = 1, \dots, N; k = 1, \dots, K \tag{2.7}$$

Tujuan dari model ini adalah untuk meminimalkan total biaya *travel*. Kendala (2.2) membatasi bahwa total jumlah *demand* yang dibawa oleh kendaraan k tidak boleh melebihi kapasitas dari kendaraan tersebut. Kendala (2.5) untuk menunjukkan bahwa tiap konsumen hanya dapat dilayani oleh satu kendaraan saja. Kendala (2.6) dan (2.7) digunakan untuk memastikan bahwa tiap konsumen dikunjungi oleh kendaraan yang sama dengan yang sudah dijadwalkan untuk konsumen tersebut (Sutapa *et al*, 2003).

2.1.3 Jenis-Jenis VRP

Dalam penggunaan VRP untuk dunia nyata, banyak faktor sampingan yang muncul. Faktor-faktor tersebut berpengaruh pada munculnya variasi dari VRP, antara lain (Prana,2007):

1. *Capacitated* VRP (CVRP) adalah sebuah VRP dimana diberikan sejumlah kendaraan dengan kapasitas tersendiri yang harus melayani sejumlah permintaan pelanggan yang telah diketahui untuk satu komoditas dari sebuah depot dengan biaya transit minimum.
2. VRP *with Time Windows* (VRPTW) hampir sama dengan VRP namun VRPTW memiliki batas tambahan yaitu sebuah jangka waktu, yang berhubungan dengan setiap pelanggan $v \in V$, yang mendefinisikan sebuah jangka waktu $[ev, lv]$ dimana sang pelanggan harus disuplai.
3. *Mulitple Depot* VRP (MDVRP) yaitu misalkan sebuah perusahaan mungkin memiliki lebih dari satu depot. Jika pelanggan-pelanggannya terkumpul di sekitar depot-depot yang ada, maka masalah pendistribusiannya harus dimodelkan menjadi sebuah kumpulan dari VRP-VRP yang independen. Namun, jika pelanggan dan depot-depot yang ada saling bercampur aduk (tidak terkumpul secara teratur, bisa ada satu pelanggan dilayani lebih dari satu depot atau sebaliknya) maka masalahnya menjadi *Multi-Depot Vehicle Routing Problem.*
4. VRP *with Pick-Up and Delivering* (VRPPD) VRPPD adalah sebuah VRP dimana ada peluang kejadian pelanggan mengembalikan barang yang sudah diantarkan. Dalam VRPPD perlu diperhatikan bahwa barang yang dikembalikan dapat dimasukkan ke dalam kendaraan pengantar.

5. *Split Delivery* VRP (SDVRP) adalah perluasan VRP jika tiap pelanggan dapat dilayani dengan kendaraan yang berbeda andaikan biayanya dapat berkurang. Perluasan ini perlu dilakukan jika jumlah permintaan pelanggan sama besar dengan kapasitas dari kendaraan. SDVRP memiliki tujuan meminimalisasi jumlah kendaraan dan total waktu perjalanan untuk pelayanan.
6. *Stochastic* VRP (SVRP) adalah variasi VRP yang terjadi jika faktor sampingan yang muncul bersifat acak. Ada tiga bentuk SVRP, yaitu pelanggan *stochastic* adalah tiap pelanggan v_i ada memiliki peluang p_i dan tidak ada dengan peluang $1 - p_i$, permintaan *stochastic* adalah jumlah permintaan di untuk tiap pelanggan adalah variabel random, dan waktu *stochastic* yaitu waktu pelayanan δ_i dan waktu pelayanan t_{ij} adalah variabel random.
7. *Periodic* VRP. Dalam *Periodic Vehicle Routing Problem* atau PVRP ini, VRP digeneralisasi dengan memperluas rentang perencanaan pengiriman menjadi *M* hari, dari semula hanya dalam rentang sehari. PVRP memiliki tujuan meminimalisasi jumlah kendaraan dan total waktu perjalanan untuk melayani tiap pelanggan.

2.2 Algoritma *Nearest Neighbor*

Pop *et al*. (2011) mendefinisikan algoritma *nearest neig*hbor merupakan teknik yang sederhana dan terbuka untuk berbagai macam variasi masalah. Pada algoritma ini, peraturannya hanya pergi ke node terdekat yang belum dikunjungi dengan mengikutkan beberapa batasan. Cara penggunaanya dimulai dengan berangkat dari depot, setiap node dikunjungi hanya satu kali dan menjumlahkan waktu tur sementara dengan tidak melebihi kapasitas kendaraan. Apabila jumlah waktu tur sementara melebihi kapasitas kendaraan, maka kendaraan akan memulai lagi dari depot dan mengunjungi node yang belum didatangi. Apabila semua node sudah dikunjungi, maka algoritma dihentikan. Berdasarkan hasil data yang dikumpul-kan, maka setiap node akan dilewati tepat satu kali dan menciptakan konstruksi rute berdasarkan hasil dari algoritma.

2.3 Algoritma *Tabu Search*

2.3.1 Pengertian umum

Kata *tabu* atau *taboo* berasal dari bahasa Tongan yaitu salah satu bahasa Polynesia yang digunakan oleh penduduk pribumi dari pulau Tonga untuk mengungkapkan sesuatu yang tidak boleh disentuh karena merupakan sesuatu yang keramat.

Lebih rinci lagi, *tabu search* berdasarkan premis yang bersifat *problem solving* atau memecahkan masalah, untuk dikualifikasikan cerdas, harus menyertakan *adaptive memory* dan *responsive exploration*. Fitur *adaptive memory* dan *responsive exploration* dalam *tabu search* membuat implementasi prosedur yang dapat melakukan pencarian berbagai solusi secara ekonomis dan efektif. Karena pilihan-pilihan lokal dipandu dengan informasi yang dikumpulkan selama pencarian *tabu search* sangat berbeda dibandingkan dengan pola tanpa memori (*memoriless*) yang sangat bergantung pada proses semi acak yang mengimplementasikan sebuah bentuk sampling. Contoh dari metode tanpa memori adalah *heuristic greedy*, dan pendekatan *annealing* dan *genetic* terinspirasi oleh metafor fisika dan biologi. *Adaptive memory* juga berbeda dengan desain memori yang kaku pada algoritma *branch and bound*.

Upaya untuk melakukan *responsive exploration* dalam *tabu search*, baik itu implementasi *deterministic* atau *probabilistic*, berasal dari pemahaman bahwa suatu pilihan strategi yang buruk dapat menghasilkan informasi yang lebih banyak dibandingkan suatu pilihan acak yang baik. Dalam suatu sistem yang menggunakan memori sebuah pilihan buruk yang berdasarkan strategi dapat memberikan petunjuk yang bermanfaat tentang bagaimana strategi tersebut dapat diubah menjadi lebih baik.

Responsive exploration mengintegrasikan prinsip-prinsip dasar dari *intelegence search*, seperti memanfaatkan fitur solusi yang baik saat menjelajahi area baru yang menjanjikan. *Tabu search* memperhatikan pencarian cara baru yang lebih efektif dalam memperoleh keuntungan dari mekanisme yang berhubungan dengan *adaptive memory* dan *responsive exploration*. Pengembangan pola baru dan kombinasi-kombinasi strategi membuat tabu suatu area yang luas untuk penelitian dan studi empiris (Glover dan Laguna, 1997).

2.3.2 Penggunaan memori

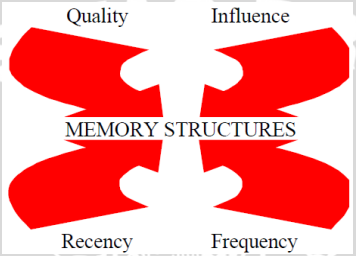
Struktur memori dalam *tabu search* menggunakan empat prinsip utama yaitu (Berlianty dan Arifin, 2010):

1. *Recency* atau lebih lengkapnya *recency based memory*, memori yang tetap menjaga struktur terbaik dari solusi awal yang ditemukan selama proses pencarian pada setiap iterasinya, sehingga apabila pada suatu iterasi ditemukan struktur/solusi yang lebih baik maka solusi ini akan tetap dipertahankan sampai ditemukannya solusi baru yang lebih baik lagi. Agar *recency* dapat bekerja dengan baik, maka didukung oleh *tabu list*, dan dalam *tabu list* ini terdapat *tabu active*. *Tabu list* merupakan suatu set memori yang memberikan informasi tentang struktur dari solusi awal yang pernah diambil, sehingga *tabu list* akan menghindari digunakannya struktur yang sama untuk menghasilkan suatu solusi jika struktur tersebut pernah dipakai. Sedangkan *tabu active* adalah *tabu list* yang berada paling akhir, artinya bahwa solusi itu merupakan solusi dari iterasi yang paling akhir yang sedang dikunjungi. Untuk membatasi *range* atau jumlah dari *tabu list* sendiri dinamakan dengan *tabu tenure*.
2. *Frequency* menyediakan sebuah tipe informasi yang merupakan kumpulan informasi yang telah direkam oleh *recency based memory*. Sehingga *frequency* dan *recency* dapat saling melengkapi untuk membentuk suatu informasi permanen guna mengevaluasi pergerakan/*move* yang terjadi.
3. *Quality* adalah kemampuan untuk membedakan solusi terbaik yang dikunjungi selama pencarian atau iterasi berlangsung.
4. *Influence* mempertimbangkan efek yang terjadi dari pemilihan solusi yang dipilih selama pencarian berlangsung, tidak hanya kualitas saja dipertimbangkan melainkan juga strukturnya.

Memori yang digunakan dalam *tabu search* bersifat eksplisit dan juga atributif. Memori eksplisit merekam seluruh solusi, teutama terdiri dari solusi penting yang dikunjugi selama pencarian. Suatu perluasan dari memori ini merekam solusi penting yang sangat atraktif namun merupakan solusi tetangga yang belum tereksplorasi.

Sebagai alternatif, *tabu search* menggunakan memori atributif untuk tujuan sebagai panduan. Jenis memori ini merekam informasi tentang atribut-atribut solusi yang mengalami perubahan dalam proses perpindahan dari satu solusi ke solusi yang lain. Sebagai

contoh, dalam suatu grafik atau jaringan, atribut dapat terdiri dari nodes atau arah yang ditambahkan, dihilangkan atau direposisi dengan mekanisme perpindahan. Dalam penjadwalan produksi, daftar digunakan sebagai atribut untuk mencegah atau mendorong metode untuk mengikuti arah pencarian tertentu (Glover dan Laguna, 1997). Empat prinsip utama dalam stuktur memori *tabu search* disajikan pada Gambar 2.2.



Gambar 2.2 Struktur memori *tabu search* (Glover dan Laguna, 1997)

Tabu search dimulai dengan membuat solusi acak dan secara berturut-turut pindah ke salah satu tetangganya. Setiap kali pergerakan dilakukan, solusi sebelumnya akan dimasukkan ke dalam suatu daftar yang disebut *tabu list*. Dari suatu solusi yang diberikan, tidak semua tetangganya dapat dicapai. Setiap perpindahan akan membawanya pada solusi terbaik yang berada disekitarnya, tetapi jika perpindahan itu ada di *tabu list* maka hanya akan diterima apabila dapat menurunkan nilai dari fungsi obyektifnya sampai di bawah level yang telah dicapai sejauh ini *(aspiration level)* (Gendreau *et al*, 1994).

Algoritma dasar *tabu search* adalah sebagai berikut (Setiawan dan Suparno,2009):

1. Langkah 1: memilih sebuah solusi inisal i dalam S. Set $i^*=i$ dan $k=0$.
2. Langkah 2: Set $k=k+1$ dan membangkitkan subset V^* dari solusi dalam $N(i,k)$
3. Langkah 3: Memilih sebuah *best j* dalam V^* dan set $i=j$
4. Langkah 4: Jika $f(i) < f(i^*)$ maka set $i^* = i$
5. Langkah 5: Meng-*update tabu* dan kondisi tujuan
6. Langkah 6: Jika *stopping condition* telah didapatkan maka *stop*. Jika tidak, mengulangi langkah 2.