PERBANDINGAN PENGGUNAAN ALGORITMA GENETIKA DENGAN ALGORITMA KONVENSIONAL PADA TRAVELING SALESMAN PROBLEM

Mohamad Irvan Faradian – NIM : 13504024

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : itb@students.itb.ac.id, if14024@students.if.itb.ac.id

Abstrak

Persoalan TSP (*Traveling Salemans Problem*) atau pedagang keliling ialah sebuah persoalan optimasi untuk mencari rute terpendek bagi seorang pedagang keliling (*salesman*) yang ingin menjajakan produknya di beberapa kota dengan batasan bahwa dia pergi dari sebuah kota ke setiap kota-kota lainnya yang menjadi target penjualan produknya dan harus kembali ke kota asal keberangkatan. Persoalan optimasi yang ingin dicapai ialah rute yang dilalui dan biaya yang digunakan paling minimum. Bila dipandang dari sudut komputasinya, algoritma ini dapat diselesaikan dengan cepat walaupun dengan menggunakan algoritma *brute force* sekalipun, jika kota-kota yang akan dikunjunginya sedikit. Namun, jika kota-kota yang akan dikunjungi banyak, maka algoritma seperti *Brute Force* tidaklah menjadi pilihan lagi. Sebab, Algoritma *Brute Force* sendiri memiliki kompleksitas O(n!) jika kota-kota yang akan dikunjungi diasumsikan sebagai graf lengkap yang saling terhubung antara satu kota dengan kota yang lainnya. Beberapa metode telah dikembangkan untuk memecahkan persoalan ini namun belum ditemukan algoritma penyelesaian yang optimal. Salah satu algoritma yang muncul untuk menyelesaikan persoalan ini ialah Algoritma Genetika yang terinspirasi dari Teori Evolusi Darwin.

***Kata kunci**: Traveling Salesman Problem, Algoritma Genetika*

1. Pendahuluan

Persoalan *Traveling Salesman Problem* merupakan salah satu persoalan kombinatorial. Banyak permasalahan yang dapat direpresentasikan dalam bentuk *Traveling Salesman Problem*. Persoalan ini sendiri menggunakan representasi graf untuk memodelkan persoalan yang diwakili sehingga lebih memudahkan penyelesaiannya. Diantara permasalahan yang dapat direpresentasikan dengan TSP ialah masalah transportasi, efisiensi pengiriman surat atau barang, perancangan pemasangan pipa saluran, proses pembuatan PCB (*Printed Cirtcuit Board*) dan lain-lain. Persoalan yang muncul ialah bagaimana cara mengunjungi simpul (*node*) pada graf dari titik awal ke setiap titik-titik lainnya dengan bobot minimum. Bobot ini sendiri dapat mewakili berbagai hal, seperti biaya, jarak, bahan bakar, waktu, dan lainnya.

Beberapa metode algoritma telah dikembangkan untuk menyelesaikan persoalan TSP ini. Bila dipandang dari sudut komputasinya, algoritma ini dapat diselesaikan dengan cepat walaupun dengan menggunakan algoritma brute force sekalipun, jika kota-kota yang akan dikunjunginya sedikit. Namun, jika kota-kota yang akan dikunjungi banyak, maka algoritma seperti brute force tidaklah menjadi pilihan lagi. Sebab, algoritma brute force sendiri memiliki kompleksitas O(n!) jika kota-kota yang akan dikunjungi diasumsikan sebagai graf lengkap yang saling terhubung antara satu kota dengan kota yang lainnya. Beberapa metode telah dikembangkan untuk memecahkan masalah ini namun belum ditemukan algoritma penyelesaian yang optimal. Salah satu algoritma yang muncul untuk memecahkan masalah ini ialah Algoritma Genetika yang terinspirasi dari teori evolusi Darwin.

2. Sejarah Permasalahan TSP

Permasalahan matematik yang berkaitan dengan *Traveling Salesman Problem* mulai muncul sekitar tahun 1800-an. Masalah ini dikemukakan oleh dua orang matematikawan, yaitu Sir William Rowan Hamilton yang berasal dari Irlandia dan Thomas Penyngton Kirkman yang berasal dari Inggris. Diskusi mengenai awal studi

dari persoalan TSP ini dapat ditemukan di buku *Graph Theory* 1736-1936 by N.L. Biggs, E.K. LLoyd, and R.J. Wilson, Clarendon Press, Oxford, 1976.

Bentuk umum dari persoalan TSP pertama kali dipelajari oleh para matematikawan mulai tahun 1930-an oleh Karl Menger di Vienna dan Harvard. Persoalan tersebut kemudian dikembangkan oleh Hassler Whitney dan Merril Flood di Princeton. Penelitian secara mendetail hubungan antara Menger dan Whitney, dan perkembangan persoalan TSP sebagai sebuah topik studi dapat ditemukan pada tulisan Alexander Schriver "*On the history of combinatorial optimization (till 1960)*".

3. Contoh Perkembangan Persoalan yang Muncul

Code program komputer yang dibuat untuk menyelesaikan persoalan TSP telah berkembang semakin baik dari tahun ke tahun. Tanda yang paling mencolok dari perkembangan metode untuk menyelesaikan persoalan TSP ialah bertambahnya jumlah simpul (*node*) yang dapat diselesaikan, mulai dari solusi persoalan 49 kota yang kembangkan oleh Dantzig, Fulkerson, dan Johnson's pada tahun 1954 sampai kepada solusi persoalan 24.978 kota 50 tahun kemudian.

a. 1954 (Dantzig49)

   Dantzig49 adalah persoalan yang diteliti oleh Dantzig, Fulkerson, dan Johnson yang dapat kita lihat dalam tulisan yang mereka buat pada tahun 1954 tentang solusi dari persoalan TSP yang terdiri dari 48 negara bagian di Negara Amerika ditambah dengan Kota Washington, D. C. Para penulis kemudian bekerja dengan 49 kota tersebut. Jalur yang optimal dari 49 kota teresbut, menggunakan jalan pintas yang ada pada 7 kota selain 49 kota tersebut. Jalur yang dibuat berdasarkan jarak pada setiap jalur dalam kota di antara 49 kota tersebut. Para penulis mengatakan bahwa mereka mendapat tabel jarak dari Bernice Brown karyawan Perusahaan Rand.

b. 1971 (64 random points)

c. 1975 (67 random points)

d. 1977 (GR120)

   Gr120 telah menjadi pengujian standar bagi permasalahan TSP sejak tahun 1977. Gr120 memiliki 120 titik yang terdiri dari jarak tempuh di antara 120 kota yang terdapat di sekitar Negara Jerman. Daftar kota-kota tersebut terdapat pada Atlas Umum Negara Jerman tahun1967/1968.

e. 1980 (Lin318)

   Lin318 muncul pada tahun 1973 dalam tulisan yang dibuat oleh S. Lin dan B. W. Kernighan. Data dari Lin318 yang terdiri dari 318 tempat yang muncul dari hasil pengeboran, dimana bornya ialah sebuah sinar laser. Lin dan Kernighan menulis bahwa mereka mendapatkan datanya dari R. Haberman. Walaupun persoalan ini telah muncul pada tahun 1973, namun baru berhasil diselesaikan pada tahun 1980 oleh H. Crowder dan M. W. Padberg.

f. 1987 (Att532)

   Att532 muncul pada tahun 1987 di dalam tulisan yang dibuat oleh M. Padbert dan G. Rinaldi. Dalam Att532, terdapat data mengenai 532 kota yang berlokasi di Benua Amerika. Kemudian Padberg dan Rinaldi menulis bahwa mereka mendapatkan persoalannya dari Shen Lin karyawan dari Laboratorium AT & T Bell.

g. 1987 (Gr666)

   Gr666 pertama kali diselesaikan oleh O. Holland dan M. Groetschel yang muncul dalam Tesis PhD Olaf Holand's di tahun 1987. Data terdiri dari 666 kota menarik yang tersebar di seluruh dunia.

h. 1987 (Pr2392)

   Pr2392 memiliki data sebanyak 2.392 titik hasil kontribusi dari M. Padberg dan Rinaldi. Layout dari titik- titik tersebut dibuat oleh Perusahaan Tektronics.

i. 1994 (Pla7397)

   Pla7397 ialah sebuah programmed logic array application yang terdiri dari 7.397 kota.

j. 1998 (Usa13509)

k. 2001 (D15112)

   D15112 memiliki data mengenai 15.112 kota yang terdapat di Negara Jerman.

l. 2004 (Sw24798)

Sw24798 terdiri dari 24.978 kota yang terdapat di Negara Swedia. Data-data tersebut didapat dari National Imagery and Mapping Agency, sebuah database nasional yang menyimpan nama-nama fitur geografi.

4. Perkembangan Penyelesaian Persoalan TSP

Bagaimana kita mengukur kemajuan dalam penyelesaian persoalan TSP? Sebuah penilian sederhana pasti mengatakan bahwa metode A lebih baik daripada metode B jika A membutuhkan waktu yang lebih singkat atau sumber daya yang lebih sedikit untuk menyelesaikan setiap contoh persoalan. Ini merupakan sebuah aturan yang jelas. Namun, masalah penilaian untuk metode ini akan sulit untuk dilakukan karena metode-metode yang sangat berkaitan erat satu sama lain tidak dapat dinilai hanya melalui perbandingan yang sederhana. Sepertinya perlu dilakukan pertimbangan ulang untuk menentukan kriteria perbandingan antar metode tersebut.

Oleh karena itu, penilaian yang lebih baik harus mengesampingkan hasil contoh-contoh kasus kecil seperti di atas karena contoh-contoh kasus tersebut dapat diselesaikan oleh seluruh teknik penyelesaian yang baik. Sejauh ini, jika diberikan sejumlah n kota, penilaian seharusnya difokuskan pada n-kota yang benar-benar sulit untuk diselesaikan dengan menggunakan metode-metode yang diajukan yang nantinya akan diuji mana yang lebih baik. Dengan pendekatan ini, kita kemudian dapat menentukan apakah metode A lebih baik daripada metode B jika diberikan persoalan n-kota dengan n sebuah bilangan yang besar.

Agar ide perbandingan metode-metode di atas dapat diaplikasikan, kita dapat menganalisis metode-metode penyelesaian yang diberikan untuk dapat memberikan jaminan bahwa setiap n akan memakan waktu sejumlah f(n) untuk berapapun n-kota TSP, dimana f(n) ialah sebuah fungsi yang menghasilkan waktu yang dibutuhkan untuk menyelesaikan persoalan TSP n-kota. Sekarang untuk membandingkan dua buah metode penyelesaian, kita membandingkan fungsi mana yang menghasilkan hasil yang terbaik yang diberikan di antara dua buah solusi penyelesaian tersebut. Hal ini tentu saja menghasilkan hasil perhitungan yang salah karena sebuah metode penyelesaian yang benar-benar baik namun dianalisis dengan buruk akan terlihat buruk jika dibandingkan dengan metode penyelesaian lain yang dianalisis dengan baik. Pada beberapa persoalan komputasi, bagaimanapun juga, studi mengenai algoritma + fungsi telah memberikan hasil yang baik yang penting bagi pengembangan untuk penyelesaian persoalan praktis - hal ini telah menjadi subjek studi utama di dalam Bidang Ilmu Komputer.

Jadi, apa yang dapat kita katakan dari metode-metode penyelesaian persoalan TSP? Tentu saja mudah untuk mengembangkan metode penyelesaian yang memiliki fungsi yang memiliki f(n) = (n-1)! = (n-1) x (n-2) x (n-3) … x 3 x 2 x 1 dan jumlah jalur perjalanan antar kota yang mungkin terjadi ialah (n-1)!/2. Hasil jauh yang lebih telah dikembangkan pada tahun 1962 oleh Michael Held dan Richard Karp, yang menemukan algoritma yang menghasilkan f(n) yang memiliki proporsi $n^2 2^n$, yaitu n x n x 2 x 2 x 2 x … x 2, dimana ada sebanyak n perkalian 2. Untuk setiap n yang bernilai besar, fungsi f(n) Held-Karp akan selalu lebih kecil jika dibandingkan dengan (n-1)!.

Bagi setiap orang yang tertarik untuk menyelesaikan persoalan TSP yang besar, ada sebuah kabar buruk bahwa selama 45 tahun sejak Held dan Karp menemukan fungsi $f(n) = n^2 2^n$ ternyata tidak ditemukan fungsi f(n) yang lebih baik. Hal ini tentu saja mengecewakan karena dengan n = 30 fungsi f(n) Held-Karp menghasilkan nilai yang sangat besar. Dan untuk n = 100, adalah suatu hal yang mustahil untuk menyelesaikan persoalan ini dengan kemampuan yang dimiliki komputer yang ada saat ini.

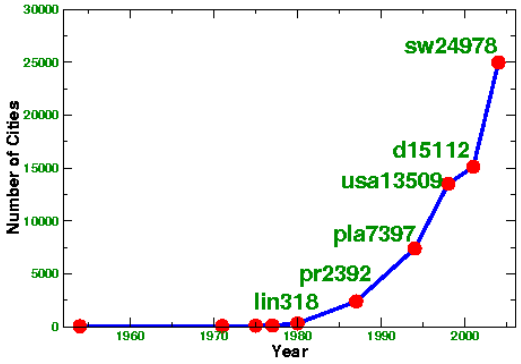
Perkembangan fungsi f(n) dalam TSP yang sangat lambat ini mungkin memang tidak dapat kita hindari; dengan kemampuan komputer yang ada saat ini, bisa jadi memang tidak ada metode penyelesaian persoalan yang menghasilkan f(n) memiliki tingkat performasi yang baik, misal n^c dimana c ialah sebuah konstanta, oleh karena itu, n x n x n x … x n dimana n muncul sebanyak c kali. Diskusi mengenai teknis permasalahan ini dapat dilihat pada tulisan Stephen Cook's dan Institut Matematika Clay menawarkan hadiah sebesar satu juta US dolar bagi siapa pun yang dapat menemukan metode yang lebih baik.

Persoalan kompleksitas TSP ialah sebuah pertanyaan yang mendalam dalam bidang matematik. Tetapi situasinya ialah sekarang kita mendapatkan sedikit informasi yang berguna dengan melihat pada kasus dengan tingkat performansi terburuk dari metode penyelesaian masalah TSP.

Dengan sedikit pilihan baik yang tersedia, para peneliti TSP telah berusaha untuk mengukur perkembangan dengan cara melihat bagaimana implementasi pada komputer dari metode-metode penyelesaian tersebut menyelesaikan persoalan pada contoh kasus yang diberikan. Maksudnya ialah bahwa dengan memperbesar ukuran dan variasi contoh kasus yang dapat diselesaikan, kita akan memperoleh kemajuan pada solusi praktis dari TSP. Walaupun pergantian proses perbandingan untuk mendapatkan perbandingan yang baik seperti yang kami tawarkan di awal lemah, tes komputasi praktis ini telah membawa para peneliti TSP kepada metode penyelesaian TSP kepada perkembangan yang jauh lebih baik. Dan yang lebih penting lagi, usaha-usaha tersebut telah mengarahkan penelitian ke dalam pengembangan alat optimasi yang bersifat umum.

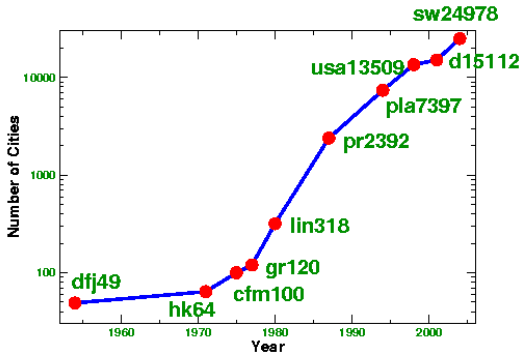
Contoh kasus yang paling umum digunakan dalam studi komputasi sekarang ialah himpunan data tes TSPLIB Gerd Reinelt. TSPLIB memiliki lebih dari 100 contoh kasus mulai dari industri, geografi, dan akademi. Untuk melengkapi koleksi ini, contoh kasus lebih jauh tersedia di koleksi National TSP dan VLSI TSP.

Sebuah tanda yang mudah dikenali untuk mengukur perkembangan pada data tes ialah perkembangan jumlah data TSP terbesar yang dapat diselesaikan yang terus meningkat selama beberapa tahun, seperti yang ditunjukkan pada gambar di bawah ini.



Catatan perkembangan di atas dimulai oleh tulisan klasik Danzig, Fulkerson, dan Johnson pada tahun 1954 dimana mereka menyelesaikan permasalahan TSP 49-kota yang terdiri dari seluruh negara bagian di Negara Amerika (Alaska dan Hawaii belum masuk di dalamnya) termasuk juga Washington, D. C. Daftar perkembangan persoalan yang muncul telah disebutkan pada pembahasan 3.1 di atas.

Perkembangan contoh persoalan TSP, diukur dengan skala logaritma terhadap jumlah kota yang dimodelkan dapat dilihat pada gambar di bawah ini.



Dari gambar di atas kita dapat melihat bahwa perkembangan penyelesaian persoalan TSP telah dicapai selama lebih dari 30 tahun ke belakang. Jika hal ini terus berlanjut, kita dapat memperkirakan bahwa untuk 30 tahun ke depan kita dapat menyelesaikan persoalan TSP dengan jumlah kota yang berjumlah jutaan. Penelitian untuk menyelesaikan permasalahan TSP dapat dilihat di http://www.tsp.gatech.edu/world/index.html.

5. Algoritma Genetika

Algoritma genetika merupakan metode penyelesaian yang terinspirasi oleh prinsip genetika dan seleksi alam yang dikemukakan oleh Darwin (Teori Evolusi Darwin). Algoritma ini digunakan untuk mendapatkan penyelesaian yang tepat untuk masalah optimasi dari satu variabel atau multivariabel. Salah satu implementasi dari Algoritma Genetika ini ialah untuk menyelesaikan persoalan TSP.

Berbeda dengan teknik pencarian konvensional, algoritma genetika bermula dari himpunan solusi yang dihasilkan secara acak. Himpunan ini disebut dengan populasi. Sedangkan setiap individu yang berada dalam populasi disebut dengan kromosom yang merupakan representasi dari solusi. Kromosom-kromosom tersebut berevolusi dalam suatu proses perulangan yang berkelanjutan yang disebut dengan generasi. Pada setiap generasi, kromosom dievaluasi berdasarkan suatu fungsi evaluasi (Gen dan Cheng, 1997). Setelah beberapa generasi, maka Algoritma Genetika akan konvergen pada kromosom terbaik, yang diharapkan merupakan solusi paling optimal (Goldberg, 1989).

Pertama kali, sebelum algoritma genetika dijalankan, maka perlu didefinisikan fungsi

fitness sebagai masalah yang akan dioptimalkan. Jika nilai *fitness* semakim besar, maka sistem yang dihasilkan akan semakin baik. Fungsi *fitness* ini ditentukan dengan menggunakan metode heuristik.

Algoritma genetika sangat tepat jika digunakan untuk menyelesaikan masalah optimasi yang kompleks dan sukar diselesaikan dengan menggunakan metode konvensional. Sebagaimana halnya proses evolusi di alam, suatu algoritma genetika yang sederhana umumnya terdiri dari tiga operasi, yaitu: operasi reproduksi, operasi *crossover* (persilangan), dan operasi mutasi. Struktur umum dari suatu algoritma genetika terdiri dari langkah-langkah:

a. Membangkitkan populasi awal secara acak.

b. Membentuk generasi baru dengan menggunakan tiga operasi di atas secara berulang-ulang sehingga diperoleh kromosom yang cukup untuk membentuk generasi baru sebagai representasi dari solusi baru.

c. Evolusi solusi yang akan mengevaluasi setiap populasi dengan menghitung nilai *fitness* setiap kromosom hingga kriteria berhenti terpenuhi. Bila kriteria berhenti belum terpenuhi, maka akan dibentuk lagi generasi baru dengan mengulangi langkah b. Beberapa kriteria berhenti yang umum digunakan ialah:

   - Berhenti pada generasi tertentu.
   - Berhenti setelah dalam beberapa generasi berturut-turut didapatkan nilai *fitness* tertinggi/terendah (tergantung persoalah) tidak berubah.
   - Berhenti bila dalam n generasi berikutnya tidak diperoleh nilai *fitness* yang lebih tinggi/rendah.

6. Penyelesaian Persoalan TSP Dengan Metode Penyelesaian Konvensional

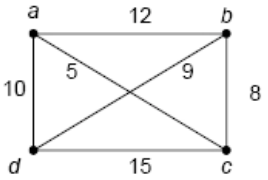
6.1. Metode Optimal

Sejak permasalahan TSP ditemukan oleh matematikawan Irlandia Sir William Rowan Hamilton dan matematikawan Inggris Thomas Penyngton Kirkman, pusat perhatian studi ini ialah menemukan secara pasti nilai minimum dari persoalan TSP dengan konsekuensi dibutuhkan waktu yang cukup lama untuk menyelesaikan persoalan TSP tersebut.

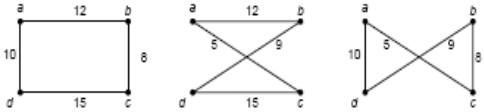
6.1.1. Algoritma *Brute Force* Dengan Complete Enumeration

Cara termudah untuk menyelesaikan TSP yaitu dengan menggunakan Algoritma *Brute Force*. Hal yang dilakukan ialah dengan cara mengenumerasi seluruh kemungkinan rute yang akan ditempuh. Setelah itu, akan dibandingkan dari seluruh kemungkinan rute yang telah dienumerasi tersebut, rute mana yang memiliki lintasan / bobot yang paling minimum.

Misalkan diberikan contoh kasus seperti di bawah ini:



Jumlah titik (tempat) yang terdapat dalam contoh kasus di atas ialah empat buah, dan jumlah kemungkinan jalur yang akan dilalui ada tiga buah, yaitu:



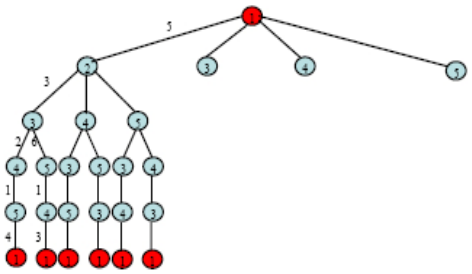
- Lintasan pertama = (a → b → c → d → a) atau (a → d → c → b → a) memiliki panjang lintasan = 10 + 12 + 8 + 15 = 45
- Lintasan kedua = (a → c → b → d → a) atau (a → d → b → c → a) memiliki panjang lintasan = 12 + 5 + 9 + 15 = 41
- Lintasan kedua = () atau () memiliki panjang lintasan = 10 + 5 + 9 + 8 = 32

Dari hasil ketiga enumerasi di atas didapatkan panjang jalur lintasan paling minimum yaitu 32. Namun, jumlah enumerasi dari algoritma ini ialah (n - 1)! yang akan memerlukan waktu yang sangat lama untuk mendapatkan panjang lintasan paling minimum jika nilai n bernilai sangat besar.

6.1.2. Algoritma *Branch and Bound*

Seperti Algoritma *Brute Force* di atas yang mengenumerasi satu per satu kemungkinan

jalur yang akan ditempuh, Algoritma *Branch and Bound* ternyata tidak memiliki kompleksitas waktu yang lebih baik dimana algoritma ini juga memiliki kompleksitas waktu (n - 1)! Contoh kasus penyelesaian persoalan TSP dengan menggunakan algoritma ini sebagai berikut:



Untuk contoh kasus di atas, panjang lintasan terpendek yang didapatkan ialah 15.

6.1.3. Dynamic Programming

Misalkan $G = (V, E)$ ialah sebuah graf lengkah berarah dengan sisi-sisi yang diberi harga $c_{ij} > 0$ untuk setiap i dan j, dimana i dan j ialah simpul-simpul yang berada di dalam V. Misalkan $|V| = n$ dan $n > 1$. Setiap simpul diberi nomor 1, 2, 3, …, n.

Asumsikan bahwa perjalanan yang akan ditempuh dimulai dan berakhir pada simpul 1.

Setiap perjalanan sendiri pasti terdiri dari sisi (1, k) untuk beberapa $k \in V - \{1\}$ dan sebuah lintasan dari simpul k ke simpul 1.

Lintasan dari simpul k ke simpul 1 tersebut melalui setiap simpul di dalam $V - \{1, k\}$ tepat hanya sekali.

Prinsip optimalisasi : jika perjalanan tersebut optimal, maka lintasan dari simpul k ke simpul 1 juga menjadi lintasan k ke 1 terpendek yang melalui simpul di dalam $V - \{1, k\}$.

Misalkan $f(i, S)$ adalah bobot lintasan terpendek yang berawal pada simpul i, yang melalui semua simpul di dalam S dan berakhir pada simpul 1.

Nilai $f(1, V - \{1\})$ adalah bobot perjalanan terpendek. Berdasarkan prinsip optimalitas tersebut, diperoleh hubungan sebagai berikut:

$$f(1, V - \{1\}) = \min_{2 \le k \le n} \{c_{1k} + f(k, V - \{1, k\})\} \quad (1)$$

Dengan merampatkan persamaan (1), diperoleh

$$f(i, \varnothing) = c_{i,1}, \ 2 \le i \le n$$

(basis)

$$f(i, S) = \min_{j \in S} \{c_{ij} + f(j, S - \{j\})\}$$

(rekurens) (2)

Persaman (1) dapat dipecahkan untuk memperoleh (1) jika kita mengetahui $f(k, V - \{1, k\})$ untuk seluruh pilihan nilai k. Nilai f tersebut dapat diperoleh dengan menggunakan persamaan (2).

Kita menggunakan persamaan (2) untuk memperoleh $f(i, S)$ untuk $|S| = 1$, kemudian kita dapat memperoleh $f(i, S)$ untuk $|S| = 2$, dan seterusnya. Bila $|S| = n - 1$, nilai i dan S ini diperlukan sedemikian sehingga $i \neq 1$, $1 \notin S$, dan $i \notin S$.

Tinjau persoalan TSP untuk n = 4.

$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

Matriks di atas ialah matriks ketetanggaan dari suatu graf yang akan kita cari lintasan terpendeknya.

- Tahap 1:
  - $f(i, \emptyset) = c_{ij}$, $2 \le i \le n$

  Diperoleh:

  $f(2, \emptyset) = c_{21} = 5$;

  $f(3, \emptyset) = c_{31} = 6$;

  $f(4, \emptyset) = c_{41} = 8$;

  terdapat tiga kemungkinan yang dapat dijadikan lintasan pertama.

- Tahap 2:

  $$f(i, S) = \min_{j \in S} \{c_{ij} + f(j, S - \{j\})\}$$

  Untuk $|S| = 1$

  Diperoleh:

  $f(2, \{3\}) = \min\{c_{23} + f(3, \emptyset)\} = \min\{9 + 6\} = \min\{15\} = 15$;

$f(3, \{2\}) = \min\{c_{32} + f(2, \emptyset)\} = \min\{13 + 5\} = \min\{18\} = 18;$

$f(4, \{2\}) = \min\{c_{42} + f(2, \emptyset)\} = \min\{8 + 5\} = \min\{13\} = 13;$

$f(2, \{4\}) = \min\{c_{24} + f(4, \emptyset)\} = \min\{10 + 8\} = \min\{18\} = 18;$

$f(3, \{4\}) = \min\{c_{34} + f(4, \emptyset)\} = \min\{12 + 8\} = \min\{20\} = 20;$

$f(4, \{3\}) = \min\{c_{43} + f(3, \emptyset)\} = \min\{9 + 6\} = \min\{15\} = 15;$

- Tahap 3:

$$f(i,S) = \min_{j \in S}\{c_{ij} + f(j, S - \{j\})\}$$

Untuk $|S| = 2$ dan $i \neq 1$, $1 \notin S$ dan $i \notin S$.

Diperoleh:

$f(2, \{3, 4\}) = \min\{c_{23} + f(3, \{4\}), c_{24} + f(4, \{3\})\}$

$= \min\{9 + 20, 10 + 15\}$

$= \min\{29, 25\} = 25$

$f(3, \{2, 4\}) = \min\{c32 + f(2, \{4\}), c34 + f(4, \{2\})\}$

$= \min\{13 + 18, 12 + 13\}$

$= \min\{31, 25\} = 25$

$f(4, \{2, 3\}) = \min\{c42 + f(2, \{3\}), c43 + f(3, \{2\})\}$

$= \min\{8 + 15, 9 + 18\}$

$= \min\{23, 27\} = 23$

Dengan menggunakan persamaan (1) diperoleh:

$f(1, \{2, 3, 4\}) = \min\{c_{12} + f(2, \{3, 4\}), c_{13} + f(3, \{2, 4\}), c_{14} + f(4, \{2, 3\})\}$

$= \min\{10 + 25, 15 + 25, 20 + 23\}$

$= \min\{35, 40, 43\} = 35$

Jadi, bobot perjalanan yang berawal dan berakhir di simpul 1 ialah 35.

Lintasan yang dilalui di dalam perjalanan tersebut dapat direkonstruksi jika kita menyimpan pada setiap f(i, S) nilai j yang meminimumkan ruas kanan dari simpul 1 selanjutnya ke simpul 2.

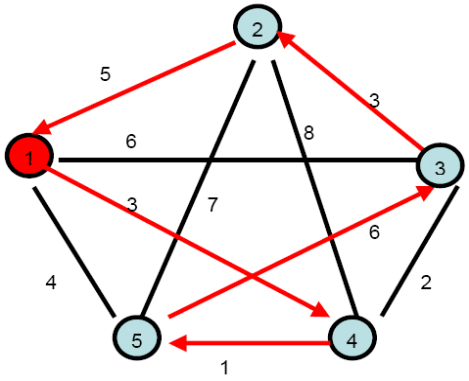
Misalkan J(i, S) ialah nilai yang dimaksudkan tersebut. Maka, J(1, {2, 3, 4}) = 2. Jadi, tur mulai dari simpul 1 selanjutnya ke simpul 2.

Simpul berikutnya dapat diperoleh dari f(2, {3, 4}) yang mana J(2, {3, 4}) = 4. Jadi, simpul berikutnya adalah simpul 4.

Simpul terakhir dapat diperoleh dari f(4, {3}), yang mana J(4, {3}) = 3. Jadi, tur yang optimal adalah 1, 2, 4, 3, 1 dengan bobot (panjang) = 35.

6.2. Metode Aproksimasi

6.2.1. Greedy Heuristic



Pada algoritma ini, pemilihan lintasan akan dimulai pada lintasan yang memiliki nilai paling minimum, setiap mencapai suatu kota, algoritma ini akan memilih kota selanjutnya yang belum dikunjungi dan memiliki jarak yang paling minimum. Algoritma ini disebut juga Nearest Neighbour.

Kompleksitas algoritma ini memang sangat mengagumkan yaitu O(n), tetapi hasil yang kita dapat bisa sangat jauh dari hasil yang optimal, semakin banyak kota semakin besar pula perbedaan hasil yang dicapai. Misalnya untuk contoh kasus yang sama dengan algoritma Branch and Bound sebelumnya yang menghasilkan nilai 15, maka algoritma ini menghasilkan nilai 18 berbeda sebesar 20% dari hasil sebelumnya padahal jumlah kota hanya 5 buah.

7. Penyelesaian TSP dengan menggunakan Algoritma Genetika

Dari seluruh algoritma yang telah disebutkan di atas untuk menyelesaikan persoalan TSP, masih ada sebuah algoritma lagi yang perlu ditinjau untuk menyelesaikan persoalan TSP. Algoritma Genetika merupakan salah satu algoritma alternatif yang dapat digunakan sebab prosesnya cepat dan memberikan hasil yang diinginkan. Selain itu, algoritma genetika juga mampu memberikan suatu solusi pada waktu kapanpun.

Bagaimana algoritma genetika dapat menyelesaikan TSP yaitu solusi direpresentasikan ke dalam suatu kromosom yang berisi dari nomor urut kota-kota selain kota asal. Masing-masing nomor urut tidak boleh muncul lebih dari satu kali di dalam kromosom sehingga satu kromosom merepresentasikan satu rute perjalanan (satu solusi) yang valid.

7.1. Pseudo-code Algoritma Genetika

Berikut ini ialah pseudo-code yang dibuat untuk menyelesaikan persoalan TSP dengan menggunakan Algoritma Genetika.

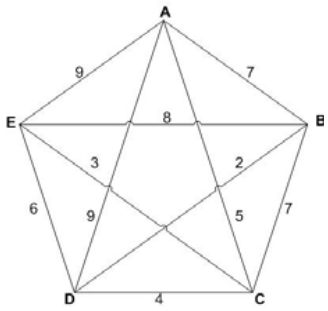
```
function Fitness (Kromosom[i]) → integer
{menghitung nilai fitness dari masing-masing
kromosom}
Deklarasi
    Jum : integer
    j : integer
    Kromosom[][] : array of integer of
    integer
function Jarak(input A, B : integer) →
integer
{menghasilkan jarak antara dua kota A dan B}
Algoritma

Jum ← Jarak(A,Kromosom[i][1])
for j←2 to 4 do
  Jum ← Jum +
  Jarak(Kromosom[i][j-1],Kromosom[i][j])
endfor
Jum ← sum + Jarak(Kromosom[i][4],A)
Return jum
```

```
procedure Crossover (input populasi: integer,
ρc: real)
{melakukan pemilihan induk pada proses crossover}
Deklarasi
  k : integer
  R[] : array of integer
function  Random (input a-b: integer) →
integer
{menghasilkan bilangan random bilangan a hingga b}
Algoritma
K ← 0
While k ≤ populasi do
  R[k] ← Random(0-1)
  if R[k] < ρc then
    pilih Kromosom[k] sebagai induk
  endif
  k ← k+1
endwhile
```

```
function  JumlahMutasi(input JumGen,
JumlahKromosom: integer, ρm: real) →
integer
{menghitung jumlah proses mutasi}
Deklarasi
  TotalGen : integer
  JumMutasi : integer
Algoritma
TotalGen ← JumGen * JumlahKromosom
ρm ← 0.2
JumMutasi ← 0.2*TotalGen
Return JumMutasi
```

7.2. Contoh Persoalan TSP yang Diselesaikan Dengan Algoritma Genetika

Berikut contoh persoalan TSP yang diselesaikan dengan menggunakan algoritma genetika. Terdapat 5 buah kota yang akan dilalui oleh seorang pedangang keliling, misalnya Kota A,B,C,D,E. Perjalanan dimulai dari kota A dan berakhir di kota A. Jarak antar kota diperlihatkan pada graf di bawah ini:



Persoalan TSP tersebut akan diselesaikan dengan menggunakan algoritma genetika. Kriteria berhenti ditentukan terlebih dahulu yaitu apabila setelah dalam beberapa generasi berturut-turut diperoleh nilai fitness yang terendah tidah berubah. Pemilihan nilai fitness yang terendah sebagai syarat karena nilai tersebut yang merepresentasikan jarak terdekat yang dicari pada persoalan TSP ini. Ada 4 kota yang akan menjadi gen dalam kromosom yaitu kota-kota selain kota asal.

a. Inisialisasi

Misalkan kita menggunakan 6 buah populasi dalam satu generasi, yaitu

Kromosom[1] = [B D E C]

Kromosom[2] = [D B E C]

Kromosom[3] = [C B D E]

Kromosom[4] = [E B C D]

Kromosom[5] = [E C B D]

Kromosom[6] = [C D E B]

b. Evaluasi Kromosom

Kemudian kita akan menghitung nilai *fitness* dari tiap kromosom yang telah dibangkitkan pada langkah a di atas.

Fitness[1] = AB+BD+DE+EC+CA
= 7 + 2 + 6 + 3 + 5 = 23
Fitness[2] = AD+DB+BE+EC+CA
= 9 + 2 + 8 + 3 + 5 = 27
Fitness[3] = AC+CB+BD+DE+EA
= 5 + 7 + 2 + 6 + 9 = 29
Fitness[4] = AE+EB+BC+CD+DA
= 9 + 8 + 7 + 4 + 9 = 37
Fitness[5] = AE+EC+CB+BD+DA
= 9 + 3 + 7 + 2 + 9 = 30
Fitness[6] = AC+CD+DE+EB+BA
= 5 + 4 + 6 + 8 + 7 = 30

c. Seleksi Kromosom

Oleh karena pada persoalan TSP yang diinginkan yaitu kromosom dengan fitness yang lebih kecil akan mempunyai probabilitas untuk terpilih kembali lebih besar maka digunakan inverse.

Q[i] = 1 / *Fitness*[i] (1)

Q[1] = 1 / 23 = 0,043

Q[2] = 1 / 27 = 0,037

Q[3] = 1 / 29 = 0,034

Q[4] = 1 / 37 = 0,027

Q[5 = 1 / 30 = 0,033

Q[6] = 1 / 30 = 0,033

Total = 0,043 + 0,037 + 0,034 + 0,027 + 0,033 + 0,033 = 0,207

Untuk mencari probabilitas kita menggunakan rumus berikut :

P[i] = *Q*[*i*] / *Total* (2)

P[1] = 0,043 / 0,207 = 0,208

P[2] = 0,037 / 0,207 = 0,179

P[3] = 0,034 / 0,207 = 0,164

P[4] = 0,027 / 0,207 = 0,130

P[5] = 0,033 / 0,207 = 0,159

P[6] = 0,033 / 0,207 = 0,159

Dari probabilitas di atas terlihat bahwa kromosom ke-1 memiliki *fitness* paling kecil oleh karena itu memiliki probabilitas untuk terpilih pada generasi selanjutnya lebih besar daripada kromosom lainnya.

Untuk proses seleksi kita menggunakan *roulette-wheel*, untuk itu kita terlebih dahulu mencari nilai komulatif dari probabilitasnya.

C[1] = 0,028

C[2] = 0,028+0,179 = 0,387

C[3] = 0,387+0,164 = 0,551

C[4] = 0,551+0,130 = 0,681

C[5] = 0,681+0,159 = 0,840

C[6] = 0,840+0,159 = 1

Proses *roulete-wheel* adalah membangkitkan nilai acak R antara 0-1. Jika **R[k]<C[k]** maka kromosom ke-k sebagai induk, selain itu pilih kromosom ke-k sebagai induk dengan syarat **C[k-1] < R[k] < C[k]**. Kita putar roulete-wheel sebanyak jumlah kromosom yaitu 6 kali (membangkitkan bilangan acak R).

R[1] = 0,314

R[2] = 0,111

R[3] = 0,342

R[4] = 0,743

R[5] = 0,521

R[6] = 0,411

Setelah itu, populasi baru akan terbentuk, yaitu:

Kromosom[1] = [2] = [D B E C]

Kromosom[2] = [1] = [B D E C]

Kromosom[3] = [3] = [C B D E]

Kromosom[4] = [5] = [E C B D]

Kromosom[5] = [4] = [E B C D]

Kromosom[6] = [6] = [C D E B]

d. *Crossover* (pindah silang)

Pindah silang pada TSP dapat diimplementasikan dengan skema *order crossover*. Pada skema ini, satu bagian kromosom dipertukarkan dengan tetap

menjaga urutan kota yang bukan bagian dari kromosom tersebut. Kromosom yang dijadikan induk dipilih secara acak dan jumlah kromosom yang dicrossover dipengaruhi oleh parameter *crossover probability* (ρc). Misal kita tentukan ρc = 25%, maka diharapkan dalam 1 generasi ada 50% (3 kromosom) dari populasi mengalami *crossover*. Pertama kita bangkitkan bilangan acak R sebanyak jumlah populasi yaitu 6 kali.

R[1] = 0,451

R[2] = 0,211

R[3] = 0,302

R[4] = 0,877

R[5] = 0,771

R[6] = 0,131

Kromosom ke-k yang dipilih sebagai induk jika R[k] < ρc. Maka yang akan dijadikan induk adalah kromosom[2], kromosom[3], dan kromosom[6]. Setelah melakukan pemilihan induk, proses selanjutnya adalah menentukan posisi *crossover*. Hal tersebut dilakukan dengan membangkitkan bilangan acak antara 1 sampai dengan panjang kromosom-1. Dalam kasus TSP ini bilangan acaknya adalah antara 1-3. Misal diperoleh bilangan acaknya 1, maka gen yang ke-1 pada kromosom induk pertama diambil kemudian ditukar dengan gen pada kromosom induk kedua yang belum ada pada induk pertama dengan tetap memperhatikan urutannya. Bilangan acak untuk 3 kromosom induk yang akan di-*crossover* :

C[2] = 2

C[3] = 1

C[6] = 2

Proses *crossover* :

Kromosom[2] = **Kromosom[2]** >< **Kromosom[3]**

= [B D E C] >< [C B D E]

= [B D C E]

Kromosom[3] = **Kromosom[3]** >< **Kromosom[6]**

= [C B D E] >< [C D E B]

= [C D E B]

Kromosom[6] = **Kromosom[6]** >< **Kromosom[2]**

= [C D E B] >< [B D E C]

= [C D B E]

Populasi setelah di-*crossover* :

Kromosom[1] = [D B E C]

Kromosom[2] = [B D C E]

Kromosom[3] = [C D E B]

Kromosom[4] = [E C B D]

Kromosom[5] = [E B C D]

Kromosom[6] = [C D B E]

e. Mutasi

Pada kasus TSP ini skema mutasi yang digunakan adalah *swapping mutation*. Jumlah kromosom yang mengalami mutasi dalam satu populasi ditentukan oleh parameter *mutation rate(*ρm*)*. Proses mutasi dilakukan dengan cara menukar gen yang dipilih secara acak dengan gen sesudahnya. Jika gen tersebut berada di akhir kromosom, maka ditukar dengan gen yang pertama. Pertama kita hitung dulu panjang total gen yang ada pada satu populasi:

Panjang total gen = jumlah gen dalam 1 kromosom ***** jumlah Kromosom (3)

= 4 * 6 = 24

Untuk memilih posisi gen yang mengalami mutasi dilakukan dengan membangkitkan bilangan acak antara 1 – Panjang total gen yaitu 1- 24. Misal kita tentukan ρm = 20 %. Maka jumlah gen yang akan dimutasi adalah = 0,2*24 = 4,8 = 5

5 buah posisi gen yang akan dimutasi, setelah diacak adalah posisi 3, 7, 10, 20, 24.

Proses mutasi :

Kromosom[1] = [D B **C** E]

Kromosom[2] = [B D **E** C]

Kromosom[3] = [C **E** D B]

Kromosom[4] = [E C B D]

Kromosom[5] = [D B C **E**]

Kromosom[6] = [E D B **C**]

Proses algoritma genetik untuk 1 generasi telah selesai. Maka nilai *fitness* setelah 1 generasi adalah:

Fitness[1] = AD+DB+BC+CE+EA

= 9 + 2 + 7 + 3 + 9 = 30

Fitness[2] = AB+BD+DE+EC+CA

= 7 + 2 + 6 + 3 + 5 = 23

Fitness[3] = AC+CE+ED+DB+BA

= 5 + 3 + 6 + 2 + 7 = 23

Fitness[4] = AE+EC+CB+BD+DA

= 9 + 3 + 7 + 2 + 9 = 30

Fitness[5] = AD+DB+BC+CE+EA

= 9 + 2 + 7 + 3 + 9 = 30

Fitness[6] = AE+ED+DB+BC+CA

= 9 + 6 + 2 + 7 + 5 = 29

Sebelumnya telah ditentukan kriteria berhenti yaitu bila setelah dalam beberapa generasi berturut-turutdiperoleh nilai *fitness* yang terendah tidah berubah. Pada 1 generasi telah terlihat bahwa terdapat nilai *fitness* terkecil yang tidak berubah. Apabila perhitungan dilanjutkan hingga ke generasi ke-N maka diyakinkan bahwa nilai *fitness* yang terendah tetap tidak akan berubah. Walaupun perhitungan cukup dijabarkan hingga generasi ke-1 saja namun solusi yang mendekati optimal telah didapatkan. Oleh karena itu, terbukti bahwa algoritma genetika dapat menyelesaikan persoalan TSP.

8. Kesimpulan

Persoalan pedagang keliling (TSP) dapat diselesaikan dengan menggunakan Algoritma Genetika. Walaupun solusi TSP yang dihasilkan oleh algoritma ini belum tentu merupakan solusi paling optimal (misalnya apabila yang dilalui sangat banyak), namun algoritma genetika akan menghasilkan solusi yang lebih optimal pada setiap generasinya. Hal tersebut terlihat dari nilai *fitness* tiap generasi. Kelebihan algoritma genetika dibandingkan metode pencarian konvensiona pada TSP yaitu pertama, solusi dapat diperoleh kapanpun karena solusi dihasilkan pada generasi ke berapapun, kedua, algoritma genetika tidak harus membutuhkan waktu yang lama karena tidak semua kemungkinan dicoba, tergantung pada kriteria berakhirnya.

9. Daftar Pustaka

[1] Munir, Rinaldi. *Matematka Diskrit*. Bandung. Informatika ITB.

[2] Munir, Rinaldi. *Strategi Algoritmik*. Bandung. Informatika ITB.

[3] http://en.wikipedia.org/wiki/Traveling_salesman_problem.html diakses tanggal 2 Januari 2007 pukul 14.12.

[4] http://www.tsp.gatech.edu/ diakses tanggal 2 Januari 2007 pukul 14.45

[6] http://en.wikipedia.org/wiki/Genetic_algorithms.html diakses tanggal 2 Januari pukul 15.30

[7] Suyanto. *Algoritma Genetika dalam MATLAB*, ANDI, Yogyakarta, 2005.

[8] Fitrah, Aulia dkk. *Penerapan Algoritma Genetika pada Persoalan Pedagang Keliling*. Bandung, Informatika ITB.