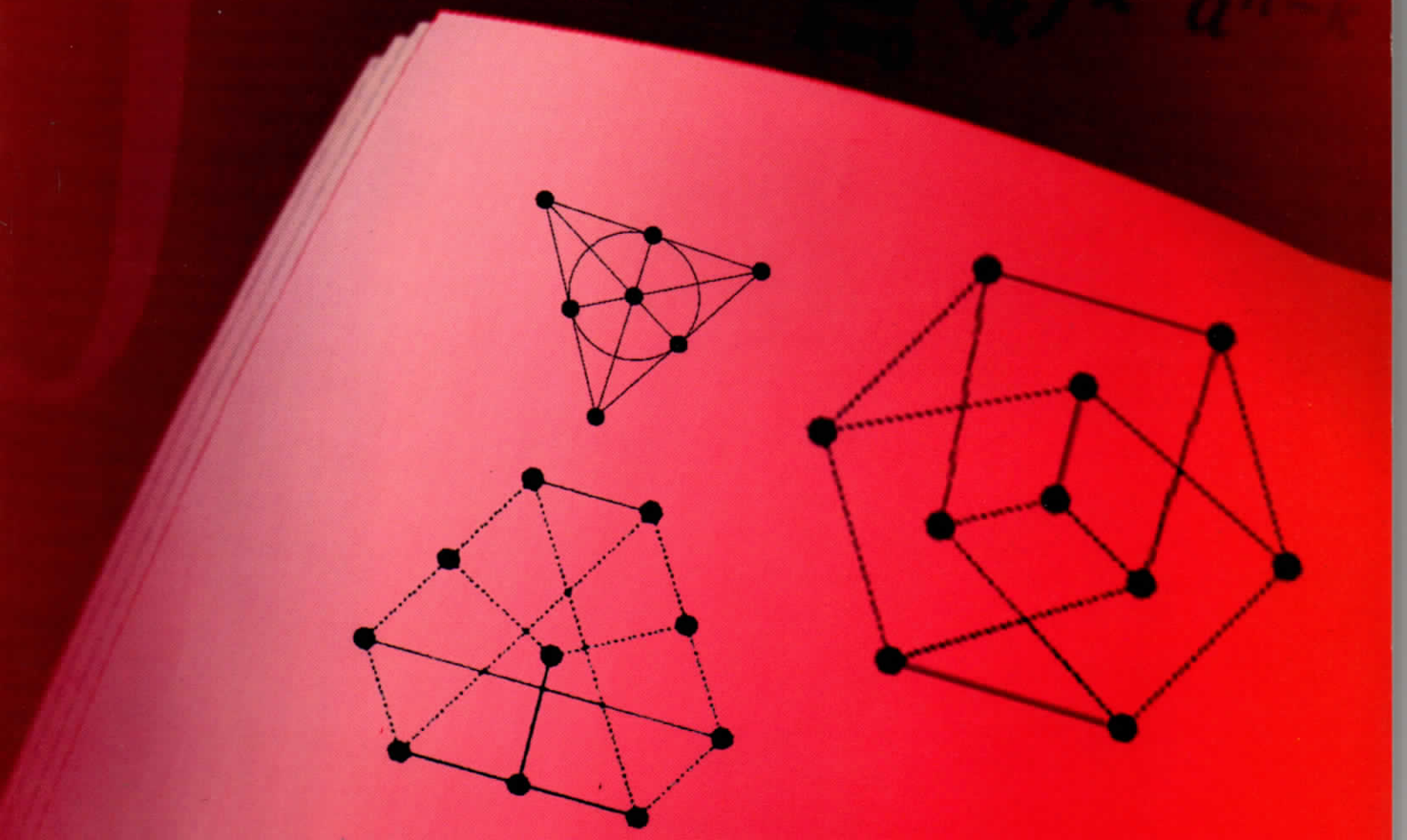
ISSN 2338-1396

JURNAL MATEMATIKA

Volume 1, Nomor 1, Januari 2013, Hlm. 1-103



JURNAL MATEMATIKA	Volume 1	Nomor 1	Halaman 1 - 103	Malang Januari, 2013	ISSN 2338-1396

ISSN 2338-1396
Volume 1, Nomor 1, Januari 2013
Halaman 1-103

JURNAL

MATEMATIKA

Terbit dua kali dalam satu volume yaitu Januari dan Juli (ISSN 2338-1396) berisi tulisan ilmiah tentang gagasan konseptual dalam bidang matematika, hasil penelitian, kajian pustaka, analisis dan aplikasi teori.

Alamat Penyunting dan Tata Usaha: FMIPA Universitas Negeri Malang, Jl. Semarang No. 5 Malang 65145, Telepon (0341) 551312 psw. 255. Langganan 2 Nomor setahun Rp. 150.000 (Wilayah Jawa, sudang dengan ongkos kirim) dan Rp. 200.000 (Wilayah Luar Jawa, sudah dengan ongkos kirim). Uang langganan dikirim ke rekening **Jurnal Matematika** nomor rekening **1229-01-002521-53-2 padaBank BRI Capem UM a.n Jurnal Matematika.**

Jurnal Matematika diterbitkan oleh Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Malang. **Dekan**: Arif Hidayat, **Wakil Dekan I**: Subandi, **Wakil Dekan II:** Imam Supeno, **Wakil Dekan III**: Fatchur Rohman. Terbit pertama kali pada tahun 2013.

Penyunting menerima sumbangan tulisan yang belum pernah diterbitkan dalam media cetak lain. Naskah diketik dengan spasi rangkap pada kertas HVS A4 (lihat Panduan untuk Penyumbang Naskah pada sampul dalam belakang). Naskah dikirim dalam bentuk *softcopy* (Word 2007 dan pdf) ke herysusanto@mat.um.ac.id. Naskah yang masuk dievaluasi oleh Penyunting Ahli, Mitra Bestari dan atau Peninjau Ahli. Penyunting berhak mengubah tulisan untuk keseragaman format dan istilah tanpa mengubah maksud dan isinya.

ISSN 2338-1396
Volume 1, Nomor 1, Januari 2013
Halaman 1-103

JURNAL
MATEMATIKA

DAFTAR ISI

IMPLEMENTASI METODE TSP DALAM PROGRAM KOMPUTER DAN ANALISIS KINERJANYA

Sapti Wahyuningsih
Darmawan Satyananda

FMIPA Universitas Negeri Malang, Jl. Semarang 5 Malang 65145, *e-mail*: sapti_wn@yahoo.co.id

Abstract: *An implementation of the tsp method on a computer program and an analysis of its performance.*One of the graph's application that is mostly used for solving various problems in daily life is Traveling Salesman Problem (TSP). Some methods which are often studied are Nearest Neighbor, Cheapest Link, Nearest Insertion, Farthest Insertion, Arbitrary Insertion, Matrix in Normal Form, Branch and Bound. This research will implement those methods in a computer programming and also does a deeper study about the necessary conditions of the graph and the comparison of each method, focuses on the properties and characteristics of the used graphs along with their advantages and weaknesses. The implementation is done by using Borland Delphi. The execution result shows that the Farthest Insertion and Branch and Bound method gives the best result among other methods.

Keywords: *Traveling Salesman Problem (TSP), Farthest Insertion method, Branch and Bound method.*

Abstrak:*Implementasi metode tsp dalam program komputer dan analisis kerjanya.* Satu dari aplikasi *graph* yang sering digunakan untuk menyelesaikan berbagai masalah di kehidupan sehari-hari adalah *Traveling Salesman Problem (TSP).* Beberapa metode yang sering digunakan adalah *Nearest Neighbor, Cheapest Link, Nearest Insertion, Farthest Insertion, Arbitrary Insertion, Matrix in Normal Form,* dan*Branch and Bound.* Penelitian ini akan menerapkan metode-metode tersebut dalam pemrograman komputer dan juga mempelajari lebih mendalam tentang syarat-syarat perlu dari graph dan membandingkan setiap metode, yang difokuskan pada sifat dan karakteristik pada penggunaan *graph* bersama dengan keuntungan dan kelemahannya. Implementasinya dikerjakan dengan menggunakan Borland Delphi. Hasil eksekusi menunjukkan bahwa metode *Farthest Insertion*dan *Branch and Bound* memberikan hasil yang lebih baik diantara metode-metode yang lain.

Kata kunci: *Traveling Salesman Problem (TSP),* metode *Farthest Insertion,* metode *Branch and Bound.*

Salah satu aplikasi teori *graph* yang banyak digunakan untuk memecahkan berbagai masalah yang terjadi di dalam kehidupan sehari-hari adalah *Traveling Salesman Problem (TSP)*(Harary, 1983). Pada awalnya*Traveling Salesman Problem(TSP)* adalah suatu masalah *salesman* keliling yang akan mengunjungi beberapa kota dan kembali ke kota awal, di mana setiap kota harus dilewati tepat satu kali dengan jarak tempuh dan biaya yang seminimum mungkin. Namun dalam perkembangannya *Traveling Salesman Problem(TSP)* banyak dimanfaatkan di berbagai bidang misalkan untuk mencari rute terpendek bus sekolah, mendesain sirkuit listrik, mencari rute terpendek suatu perjalanan, *vehicle routing problem*, dan untuk masalah penugasan.

Traveling Salesman Problem (TSP) merupakan suatu aplikasi dari sikel Hamilton. Secara teoritis, pada *graph* lengkap dengan n titik, akan terda-

pat*(n-1)!/2*sikel Hamilton (Lawer, 1987). Persoalannya adalah bagaimana cara yang mudah, cepat, dan efisien untuk mencari sikel Hamilton dengan panjang sisi paling pendek.

Selama ini sudah banyak metode *TSP* yang bisa digunakan, baik untuk *graph* lengkap maupun tidak lengkap dengan sebarang titik. Beberapa mahasiswa Jurusan Matematika FMIPA Universitas Negeri Malang dalam skripsinya telah mengkaji metode *Nearest Neighbor, Cheapest Link, Nearest Insertion, Farthest Insertion, Arbitrary Insertion, Matriks Bentuk Normal,* serta *Branch and Bound.* Penelitian ini akan mengimplementasikan berbagai metode tersebut dalam program komputer serta melakukan kajian yang lebih mendalam tentang syarat cukup *graph*nya serta perbandingan masing-masing metode, dilihat dari sifat dan karakteristik *graph* yang digunakan, kelebihan dan kekurangannya masing-masing algoritma, dan implementasi program komputernya.

KAJIAN PUSTAKA

Konsep Dasar *Graph*

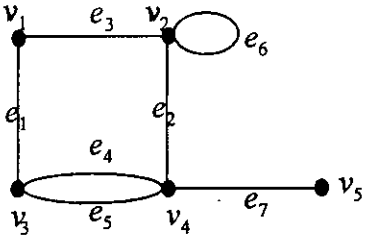
Suatu *graph* $G = (V, E)$ terdiri dari suatu himpunan tak kosong yang masing-masing unsurnya disebut titik (*vertex*), dinotasikan dengan $V(G)$, dan suatu daftar pasangan tidak terurut unsur itu yang disebut sisi(*edge*), daftar sisi dari *graphG* dinotasikan dengan $E(G)$. Jika u dan v titik-titik di G dan $e = (u, v)$ merupakan sisi di G maka e terkait dengan u dan$v$ juga dikatakan u dan *vterhubung langsung.* Banyaknya titik pada *graphG* dinotasikan dengan $|V(G)|$ dan banyaknya sisi pada *graphG* dinotasikan dengan $|E(G)|$.

Suatu *graph G* disebut *graph* terhubung jika untuk setiap pasangan titik $u, v \in V(G)$ terdapat suatu lintasan dari u ke v. *Graph* komplit (K_n) adalah *graph* dengan n titik di mana setiap pasang titik (dua titik yang berbeda) dihubungkan oleh tepat satu sisi. *Graph* komplit adalah *graph* di mana setiap dua titik yang berbeda dihubungkan oleh tepat satu sisi (Wilson dan Watkins, 1990: 36).

Suatu *graph G* disebut sebagai *graph* berbobot jika untuk setiap sisi pada *graphG* mem-punyai bobot atau nilai, bobot pada sisi (v_i, v_j) dinotasikan dengan $W(v_i, v_j)$. Dari Gambar 1 dapat dilihat bahwa $V(G) = \{v_1, v_2, v_3, v_4, v_5\}$ dan $E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$sehingga$|V(G)| = 5$ dan $|E(G)| = 7$. Sisi e_6 dikatakan sebagai *loop* karena menghubungkan suatu titik dengan titik itu sendiri.

Dari Gambar 1 dapat dilihat bahwa $V(G) = \{v_1, v_2, v_3, v_4, v_5\}$ dan $E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ sehingga$|V(G)| = 5$ dan $|E(G)| = 7$. Sisi e_6 dikatakan sebagai *loop* karena menghubungkan suatu titik dengan titik itu sendiri.

Suatu jalan(*walk*) (u, v) adalah suatu barisan $v_0\ e_1\ v_1\ e_2\ v_2 \ldots v_{n-1}\ e_n\ v_n$ dengan $v_0 = u$, $v_n = v$, v_i adalah titik, e_i adalah sisi, dan e_i menghubungkan titik v_{i-1}dan $v_i, i = 0, 1, 2, \ldots, n$. Panjang dari suatu jalan adalah banyaknya sisi yang dilewati dari titik awal sampai titik akhir.



Gambar 1. *Graph G*

Jika semua sisi (tidak perlu semua titik) pada suatu jalan adalah berbeda, maka jalan tersebut disebut *trail.* Jika semua titik adalah berbeda, maka*trail* tersebut disebut *path* (lintasan) (Wilson dan Watkins, 1990: 35).

Suatu jalan dikatakan tertutup jika titik awal sama dengan titik akhir. Jika semua sisi dari suatu jalan tertutup adalah berbeda maka jalan tersebut dinamakan *trail* tertutup. Jika semua titik pada suatu jalan tertutup adalah berbeda, maka *trail* tersebut disebut *cycle* (sikel) (Wilson dan Watkins, 1990: 35).

Sikel Hamilton pada *graph G* adalah sikel yang memuat setiap titik di G. *Graph G* disebut *graph* Hamilton (Hamiltonian) jika *graph G* memuat *sikel* Hamilton. Studi tentang eksistensi dari *graph* Hamilton telah dilakukan oleh Dirac dan Ore pada tahun 1952 dan 1960 dalam Wilson dan Watkins (1990: 146) yang menyebutkan dalam *graph* Hamilton terdapat dua syarat cukup. Dalam teoremaDiracdikatakan: misalkan G merupakan *graph* sederhana dengan n titik, $n \geq 3$. Jika derajat $v \geq n/2$ untuk setiap titik v, maka G merupakan *graph* Hamilton. Sedangkan dalam teorema Ore dikatakan: misalkan G merupakan *graph* sederhana de-ngan n titik, $n \geq 3$. Jika derajat (v) + derajat (w) $\geq n$, untuk setiap pasangan titik tidak berdekatan v dan w, maka G merupakan *Graph* Hamilton.

Untuk menyatakan keterhubungan antar titik dalam *graph* dan bobotnya, bisa digunakan matriks keterhubungan (*adjacency matrix*) atau matriks bobot (*weight matrix*). Matriks beban yang biasa dinotasikan dengan *W(G)* pada dasarnya adalah matriks berukuran $n \times n$ dengan n adalah banyaknya titik dan didefinisikan sebagai berikut.

$$W_{i,j} = \begin{cases} beban & \text{Jika ada sisi antara } v_i \text{ dan } v_j \\ 0 & \text{Jika tidak ada sisi antara } v_i \text{ dan } v_j \end{cases}$$

dengan beban menunjukkan bobot pada sisi.

Matriks keterhubungan yang dinotasikan dengan *A(G)* pada dasarnya adalah matriks berukuran $n \times n$ dengan n adalah banyaknya titik dan didefinisikan sebagai berikut.

$$W_{i,j} = \begin{cases} 1 & \text{Jika ada sisi antara } v_i \text{ dan } v_j \\ 0 & \text{Jika tidak ada sisi antara } v_i \text{ dan } v_j \end{cases}$$

Konsep Dasar *Traveling Salesman Problem*

Traveling Salesman Problem (*TSP*) didefinisikan sebagai suatu permasalahan untuk menemukan sikel Hamilton pada *graph* berbobot yang memiliki total bobot sisi minimum. Pada aplikasinya, *TSP*merupakan suatu masalah kombinatorial dalam masalah optimalisasi. Masalah tersebut dapat diuraikan secara singkat, yaitu bagaimana cara seorang *salesman* yang berkeliling untuk mengunjungi sejumlah kota, di mana ia harus mengunjungi semua kota yang ada tepat satu kali, kecuali kota asal karena pada *TSP* ini dimulai serta diakhiri pada kota yang sama, serta bagaimana agar lintasan yang dilaluinya merupakan lintasan terpendek untuk mengunjungi semua kota yang ada.

Penyelesaian dari permasalahan *TSP* ini dapat dilakukan dengan pendekatan secara langsung, yaitu dengan menghitung semua kemungkinan dari rute yang ada. Setelah dihitung total jarak dari semua rute yang ada kemudian dipilih salah satu rute yang memiliki total jarak terpendek. Jika terdapat n kota yang harus dikunjungi (termasuk kota asal) maka terdapat $(n\text{-}1)!/2$ rute yang mungkin sebagai solusi. Dengan cara tersebut maka waktu yang diperlukan dalam pencarian solusi akan semakin lama seiring dengan bertambahnya jumlah kota yang harus dikunjungi. Untuk itu dibutuhkan suatu algoritma dalam menyelesaikan permasalahan *TSP* khususnya dengan jumlah kota yang besar.

Terdapat sejumlah metode yang bisa digunakan untuk menyelesaikan permasalahan *TSP*. Metode ini terus dikembangkan oleh peneliti antara lain dapat dilihat pada Croes (1958) dan Dantzig

(1954). Pada pembahasan tulisan ini yang akan dikaji adalah metode *Nearest Neighbor, Cheapest Link, Nearest Insertion, Farthest Insertion, Arbitrary Insertion, Matriks Bentuk Normal, serta Branch and Bound.* Beberapa skripsi bim-bingan penulis yang membahas *TSP* yaitu Andriani (2005), Herdhiyanto (2005), Hasnawati (2005), dan Nurhidayah (2005).

METODE

Metode *Nearest Neighbor*

Secara sederhana metode *Nearest Neighbor* akan menentukan suatu titik sebagai titik awal, kemudian mencari sisi dengan bobot minimal yang terhubung dengan titik tersebut. Selanjutnya, dari sisi yang telah terpilih kemudian dicari sisi lain dengan bobot minimal, dan seterusnya sampai membentuk sikel Hamilton. Syarat cukup untuk metode *Nearest Neighbor* adalah *graph*nya harus lengkap.

Algoritma untuk pencarian lintasan Hamilton dengan menggunakan metode *Nearest Neighbor* adalah sebagai berikut.

1. Pilih satu titik sebagai titik awal, tambahkan ke dalam lintasan.
2. Dari titik yang terpilih, pilih titik lain yang belum terpilih dengan sisi terkait yang memiliki bobot minimum, tambahkan ke dalam lintasan. Ulangi langkah ini sampai semua titik terpilih (termuat semuanya di dalam lintasan).
3. Pilih sisi yang kembali ke titik awal untuk membentuk sikel Hamilton.

Perhatikan bahwa mungkin akan ditemui lebih dari satu sisi dengan bobot yang sama. Pada kasus seperti ini, pilih salah satu sisi lebih dulu, dan bila sudah selesai bisa diulangi dengan memilih sisi yang lain. Akibatnya akan mungkin dihasilkan lebih dari satu lintasan.

Metode *CheapestLink*

Pemilihan titik awal tidak diperlukan pada metode *Cheapest Link*tetapi dimulai dengan memilih sisi yang mempunyai bobot paling kecil, kemudian dilanjutkan dengan memilih sisi dengan bobot terkecil lain yang belum terpilih. Hal ini dilanjutkan kecuali jika sisi tersebut membentuk sikel atau dalam satu titik berderajat tiga maka dipilih sisi terkecil yang lain yang belum terpilih sampai memuat semua titik, sehingga diperoleh suatu sikel dengan bobot total minimum.

Algoritma untuk pencarian lintasan Hamilton dengan menggunakan metode *Cheapest Link* adalah sebagai berikut.

1. Pilih satu sisi dengan bobot terkecil.
2. Jika sisi tersebut membentuk sikel atau jika dalam satu titik berderajat tiga maka ulangi mengambil sisi baru yang lain yang belum terpilih dengan bobot terkecil.
3. Ulangi langkah 2 sampai semua titik terlewati dan membentuk sikel Hamilton.

Syarat cukup untuk metode *Cheapest Link* adalah *graph*nya harus lengkap. Seperti halnya metode *Nearest Neighbor*, pada metode *Cheapest Link* ada kemungkinan ditemui lebih dari satu sisi dengan bobot yang sama. Akibatnya mungkin juga akan mungkin juga akan dihasilkan lebih dari satu lintasan.

Metode *Nearest Insertion*

Metode ini berdasarkan pada *graph* lengkap. Sikel Hamilton dibangun dari sikel kecil dan secara berturut-turut ditambah dengan titik baru yang terpilih. Titik baru tersebut disisipkan pada sisi dalam sikel yang mempunyai nilai minimum. Algoritmanya bisa dituliskan sebagai berikut.

1. Pilih sembarang titik i_0 sebagai titik awal.
2. Cari titik k yang belum terpilih sehingga $C_{i\,0,k}$ adalah minimum dan membentuk subtour $i_0 - k - i_0$. Masukkan k ke dalam S.
3. Langkah seleksi: dari subtour yang telah terbentuk, cari titik k (yang tidak ada di dalam subtour) yang *terdekat* ke sembarang titik i di subtour.
4. Langkah penyisipan: cari sisi *(i, j)* dalam subtour dengan $C_{i,k} + C_{k,j} - C_{i,j}$ yang mempunyai nilai minimum. Sisipkan k diantara i dan j.
5. Kembali ke langkah 3 sampai dihasilkan sikel Hamilton.

Metode *Farthest Insertion*

Metode ini mirip dengan *Nearest Insertion*, hanya saja titik yang disisipkan adalah titik yang memiliki jarak terjauh. Sikel Hamilton dibangun dari sikel kecil dan secara berturut-turut ditambah dengan titik baru yang terpilih. Titik baru tersebut disisipkan pada sisi dalam sikel yang mempunyai nilai minimum. Algoritmanya bisa dituliskan sebagai berikut.

1. Pilih sembarang titik i_0 sebagai titik awal.
2. Caritiitik$k$ yang belum terpilih sehingga $C_{i\,0,k}$ adalah minimum dan membentuk subtour $i_0 - k - i_0$. Masukkan k ke dalam S.
3. Langkah seleksi: dari subtour yang telah terbentuk, cari titik k (yang tidak ada di dalam subtour) yang *terjauh* ke sembarang titik i di subtour.
4. Langkah penyisipan: cari sisi *(i, j)* dalam subtour dengan $C_{i,k} + C_{k,j} - C_{i,j}$ yang mempunyai nilai minimum. Sisipkan k diantara i dan j.
5. Kembali ke langkah 3 sampai dihasilkan sikel Hamilton.

Metode *heuristic* dalam penentuan TSP dikaji meskipun langkahnya lebih panjang, tetapi keoptimalan solusi dapat diandalkan. Pengkajian metode-metode *heuristic* dapat dilihat pada Bentley (1990), Christofides (1976), dan Rosenkrantz (1977).

Metode *Arbitrary Insertion Heuristic*

Metode ini mensyaratkan adanya *graph* lengkap. Bila pada *Nearest Insertion* dipilih titik yang terdekat dan pada *Farthest Insertion* dipilih yang terjauh, maka pada metode ini titik yang dipilih ditentukan sendiri (tidak melihat jaraknya). Algoritma untuk pencarian lintasan Hamilton dengan menggunakan metode *Arbitrary Insertion* adalah sebagai berikut.

1. Pilih sembarang titik i_0 sebagai titik awal.
2. Cari titik k yang belum terpilih sehingga $C_{i\,0,k}$ adalah minimum dan membentuk subtour $i_0 - k - i_0$. Masukkan k ke dalam S.
3. Langkah seleksi: dari subtour yang telah terbentuk, cari sembarang titik k (yang tidak ada di dalam subtour) yang terhubung ke sembarang titik i di subtour.
4. Langkah penyisipan: cari sisi *(i, j)* dalam subtour dengan $C_{i,k} + C_{k,j} - C_{i,j}$ yang mempunyai nilai minimum. Sisipkan k diantara i dan j.
5. Kembali ke langkah 3 sampai dihasilkan sikel Hamilton.

Metode Matriks Bentuk Normal

Metode Matriks Bentuk Normal tidak memerlukan *graph* lengkap. Metode ini didasarkan pada keterhubungan. Dipunyai matriks keterhubungan A, n adalah ordo dari matriks keterhubungan A dan h adalah bilangan bulat yang memenuhi $1 \le h \le n$,

$1 \le i \le n$ jika *entry* dari matriks A memenuhi $a(i, mod_n (i+h)) = w_{ij}$, dengan modifikasi $mod_n (n) = n$, maka matriks keterhubungan A disebut dengan matriks dalam bentuk normal. Pada matriks bentuk ini $h=1$. Lintasan normal dari suatu matriks adalah lintasan yang dibentuk dari *entrya* $(i, mod_n (i+h)) = w_{ij}$. *Anchor* dari suatu matriks adalah *entry* pada baris ke-n dan kolom ke-1 atau *entry* $[a_{n1}]$ dari suatu matriks.

Algoritma untuk pencarian lintasan Hamilton dengan menggunakan metode matriks bentuk normal adalah sebagai berikut.

1. Menentukan matriks keterhubungan langsung $A(G) = [a_{ij}]$, di mana $a_{ii} = 0$ atau *entry* pada diagonal utama sama dengan *0*, untuk $i = 1, 2, ..., n$ dari *graph* yang ada. Setiap kolom dinyatakan dengan a_i dan baris dengan b_i, dimana a_i dan b_i bersesuaian dengan v_i dan mempunyai nilai i, dimana $i = 1, 2, ..., n$.
2. Menentukan himpunan yang berisikan semua titik-titik yang terhubung dengan titik v_i.
3. Jika matriks keterhubungan langsung $A(G)$ dalam bentuk normal, maka lintasan Hamilton adalah $v_1, v_2, ..., v_k$. Jika tidak dalam bentuk normal maka dilanjutkan ke langkah selanjutnya.
4. Untuk *entry* yang sama dengan *0* pada lintasan normal maka kolom dimana terdapat *entry* yang sama dengan *0* ditukar dengan kolom yang tidak memuat *0* dan pada baris yang sama. Pencarian *entry* yang sama dengan *0* dimulai dari kiri ke kanan. Penukaran dilakukan dengan menukar kolom a_i dengan a_j kemudian menukar baris b_i dengan b_j. Penukaran dilakukan sampai lintasan normal terbentuk.
5. Jika *anchor* dari matriks $(a_{n1}) = 0$ maka *entry* tersebut diganti dengan *entry* yang tidak nol pada baris yang sama, dengan tetap memperhatikan lintasan normal.
6. Jika matriks sudah dalam bentuk normal maka titik-titik yang bersesuaian dengan $a_1, a_2, ..., a_n$ menggambarkan titik-titik yang membentuk lintasan Hamilton pada *graphG*, dimana (a_i, a_{i+1}) merupakan sisi. Jadi lintasan Hamiltonnya bersesuaian dengan kolom $a_1, a_2, ..., a_n$.

Suatu matriks keterhubungan dikatakan normal bila lintasan normal (*entry* di atas diagonal utama) bernilai tidak 0. Pada saat normal, akan didapat lintasan Hamilton.

Metode *Branch and Bound*

Metode *Branch and Bound* juga tidak memerlukan *graph* lengkap. Algoritma ini dimulai dari satu titik pada *graph* yang ditetapkan sebagai titik awal. Dari titik awal tersebut kemudian dicari titik-titik yang terhubung langsung, kemudian dihitung jaraknya (jarak parsial + jarak *underestimate*). Pilih dua titik yang mempunyai jarak minimal dengan titik awal untuk dimasukkan ke dalam *tree*, kemudian di buat lintasan parsialnya. Iterasi ini (pemilihan titik yang terhubung, penghitungan jarak, pemilihan dua titik dengan bobot terkecil, dan pembentukan lintasan parsial) terus dilakukan sampai semua titik pada *graph* termuat dalam *tree*. *Tree* yang dibentuk adalah *binary tree* dengan cabang berupa titik-titik yang mempunyai jarak minimal. *Binary tree* tersebut digunakan untuk menemukan lintasan tertutup yang memuat semua titik pada *graph* dan mempunyai bobot minimal.

Algoritma untuk pencarian lintasan Hamilton dengan menggunakan metode *Branch and Bound* adalah sebagai berikut.

1. Tentukan satu titik pada *graph* sebagai titik awal.

2. Cari titik-titik yang terhubung langsung dengan titik awal, kemudian hitung jaraknya (jarak parsial + jarak *underestimate*).
3. Pilih dua titik yang mempunyai jarak minimal dengan titik awal untuk dimasukkan ke dalam *tree*, kemudian buat lintasan parsialnya.
4. Cari lintasan parsial dengan bobot minimal dan susun kemungkinan *branch*-nya (mencari titik yang terhubung langsung dengan titik terakhir dalam lintasan parsial), kemudian hitung jaraknya (jarak parsial + jarak *underestimate*).
5. Ulangi langkah 3-4 sampai semua titik pada *graph* termuat dalam *tree*. *Tree* yang terbentuk adalah *binary tree* dengan *branch* berupa titik-titik yang mempunyai jarak minimal. *Binary tree* tersebut digunakan untuk menemukan lintasan tertutup yang memuat semua titik pada *graph* dan mempunyai bobot minimal.

HASIL DAN PEMBAHASAN

Pembahasan *TSP* terus menjadi perhatian peneliti baik dari kajian teori maupun dari kajian komputasi. Hal tersebut dapat dilihat pada artikel-artikel dari Applegate (2003), Current (1994), dan Freisleben (1996). Pada bagian ini akan dibahas mengenai hasil implementasi program dan analisis hasil program dari berbagai metode *TSP* yang telah dibuat dan diimplementasikan.

Pertimbangan Struktur Data

Secara konsep, *graph* bisa dinyatakan sebagai suatu matriks bobot atau matriks keterhubungan. Bila diimplementasikan di komputer, ada beberapa hal yang bisa dipertimbangkan dalam penentuan struktur datanya.

Permasalahannya, secara umum, lebih baik menggunakan matriks bobot daripada matriks keterhubungan karena bisa menyatakan bobot dari sisi yang menghubungkan dua titik atau bisa juga untuk menyatakan apakah ada keterhubungan antara dua titik. Jadi fungsinya sekaligus sebagai matriks keterhubungan.

Bentuk *graph*nya, *graph* yang bisa diimplementasikan dengan matriks bobot adalah *graph* yang tidak memuat sisi rangkap, karena setiap *entry* matriks hanya bisa mencatat satu nilai. Bila *graph*nya tidak berarah, maka *entry* atau elemen [i,j] akan bernilai sama dengan elemen [j,i]. Sedangkan bila *graph*nya berarah, *entry* yang berisi data menyesuaikan dengan arah yang ditunjukkan.

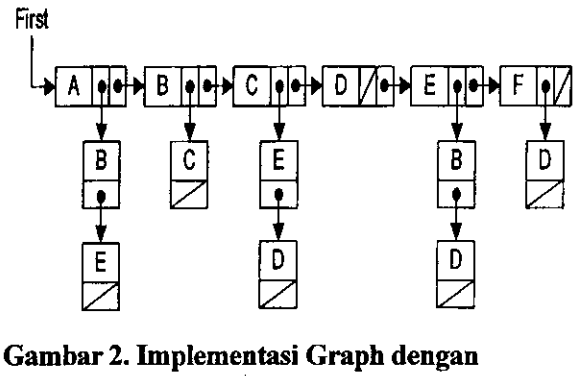
Implementasi dalam matriks (*array* 2 dimensi), baik menggunakan matriks bobot atau matriks ketetanggaan maka cara paling mudah adalah menggunakan *array* 2 dimensi. Jenis *array* yang digunakan bisa *array* statis atau *array* dinamis tergantung dari kebutuhan. Bila *graph*nya berukuran tetap, maka gunakan *array* statis dan sebaliknya, gunakan *array* dinamis. Tetapi bila program yang dibuat bisa menerima *graph* dengan ukuran berapapun maka *array* dinamis lebih tepat digunakan karena ukuran *array statis* harus sudah ditentukan pada saat program dibuat dan tidak mungkin diubah pada saat program dijalankan.

Nama-nama titik bisa menggunakan huruf atau bilangan. Bila menggunakan huruf sebagai nama titiknya, berarti maksimal ada 26 titik (dari A sampai Z). Kemungkinan lain adalah dengan menggunakan bilangan sebagai nama titiknya sehingga bisa didapat titik dengan jumlah lebih banyak. Bila menggunakan *array* dinamis maka nama titik harus menggunakan bilangan.

Implementasi dalam bentuk *linked list* (list berkait), kelemahan dari penggunaan *array* dua dimensi (matriks) adalah bila sudah ditetapkan se-

jumlah titik tetapi titik yang digunakan sangat sedikit, maka akan dipunyai *sparse matrix* karena banyak elemen matriks yang tidak terpakai. Ada cara untuk mengatasinya yaitu dengan memanfaatkan *linked list* seperti pada Gambar 2.

Dari Gambar 2 tersebut bisa diartikan bahwa ada 6 titik dalam *graph* (A sampai F), titik A terhubung dengan titik B dan titik E, titik B terhubung dengan titik C, dan titik D tidak terhubung dengan titik manapun. Dalam setiap elemen *list*, selain titik apa yang terhubung dengan suatu titik, bisa juga disisipkan informasi tentang bobot sisi yang menghubungkan keduanya. Dibandingkan dengan *array*, penggunaan list akan sedikit merepotkan dalam manipulasi keterhubungannya tetapi lebih fleksibel bila harus menyisipkan informasi di antara informasi lain yang sudah ada.



Gambar 2. Implementasi Graph dengan Menggunakan *Linked List*

Implementasi dalam bentuk array of record (array 1 dimensi), karena dengan menggunakan array 2 dimensi mungkin dihasilkan matriks yang

jarang dan bila dengan linked list akan sedikit merepotkan dalam manipulasi kaitan antar elemen list, alternatif lain adalah dengan menggunakan array 1 dimensi. Dengan cara ini hanya titik yang terhubung saja yang dicatat keberadaannya. Setiap elemen array mencatat nama atau nomor titik yang saling terhubung serta bobot antar keduanya. Bentuknya lebih baik menggunakan array dinamis karena elemennya bisa ditambah atau dihapus setiap kali diperlukan. Akan tetapi kesulitan yang muncul adalah bila kita misalnya ingin mengetahui apakah ada sisi yang menghubungkan dua titik maka kita harus menelusuri semua elemen list dari awal sampai akhir.

Implementasi Metode TSP dalam Delphi

Pada bagian ini akan dibahas bagaimana implementasi beberapa metode *TSP* dengan menggunakan bahasa Delphi.

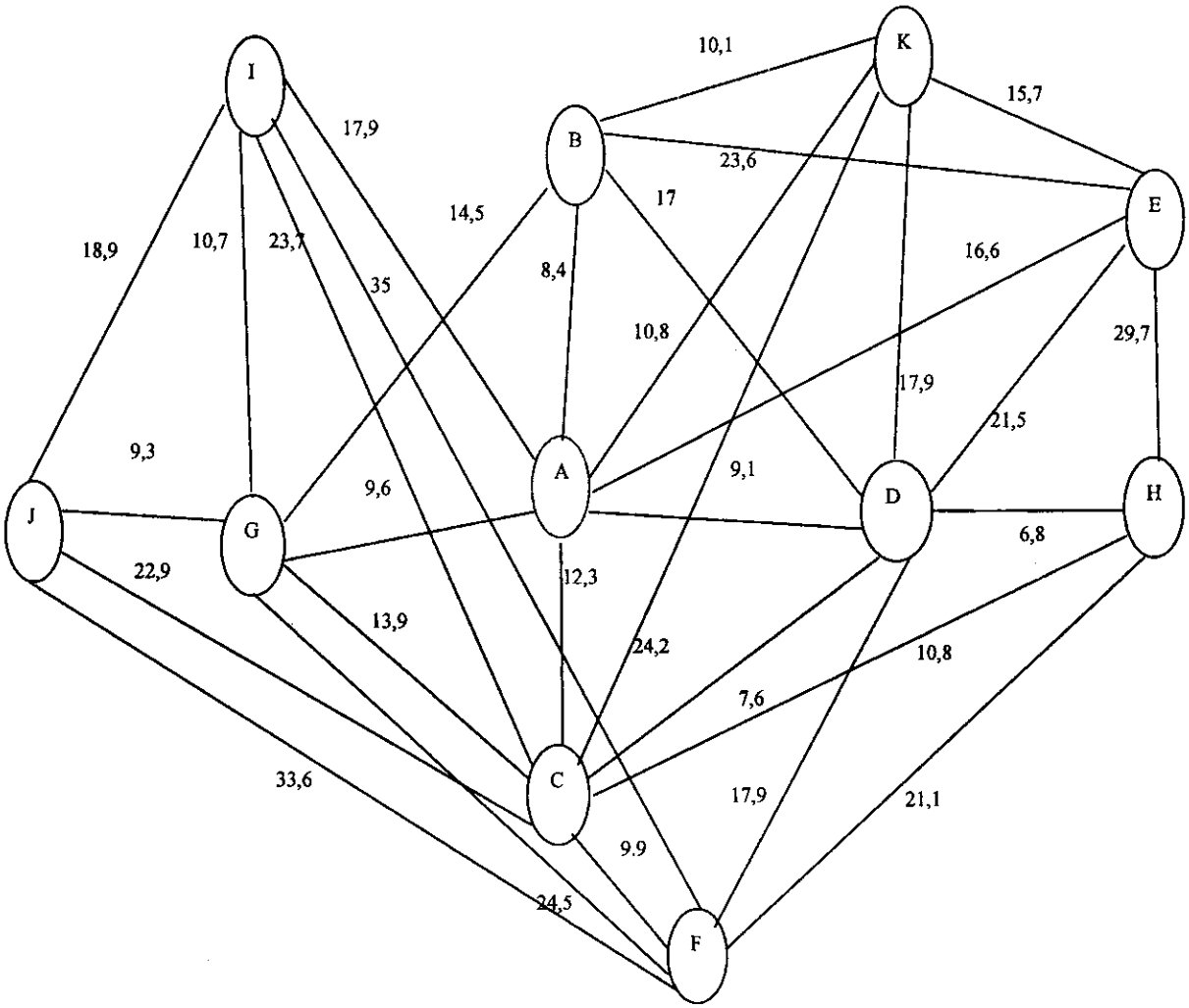
Implementasi Struktur Data

Dari ketujuh metode telah disebutkan, selanjutnya dibuat programnya dengan menggunakan Delphi. Secara umum struktur data yang digunakan adalah *array* dua dimensi yang mencatat bobot sisinya. *Array*nya berbentuk dinamis karena pada program bisa ditentukan ukuran *graph* (banyaknya titik dalam *graph*). Berikut ini deklarasi tipe data yang digunakan:

```
type
matriks=array of array of real;
recSisi=record
vAwal,vAkhir:byte;
bobot:Integer;
end;
himpunan=set of byte;
```

Matriks adalah tipe data yang digunakan untuk mencatat bobot setiap sisi (digunakan pada semua metode), yang disusun dari *array* dinamis dua dimensi bertipe real. Tipe data *rec-Sisi* digunakan untuk mencatat sisi dan bobotnya. Tipe data ini kemudian digunakan oleh variabel lainyang bertipe *TList*. Setiap elemen list yang merujuk ke variabel dinamis bertipe *rec-Sisi*.

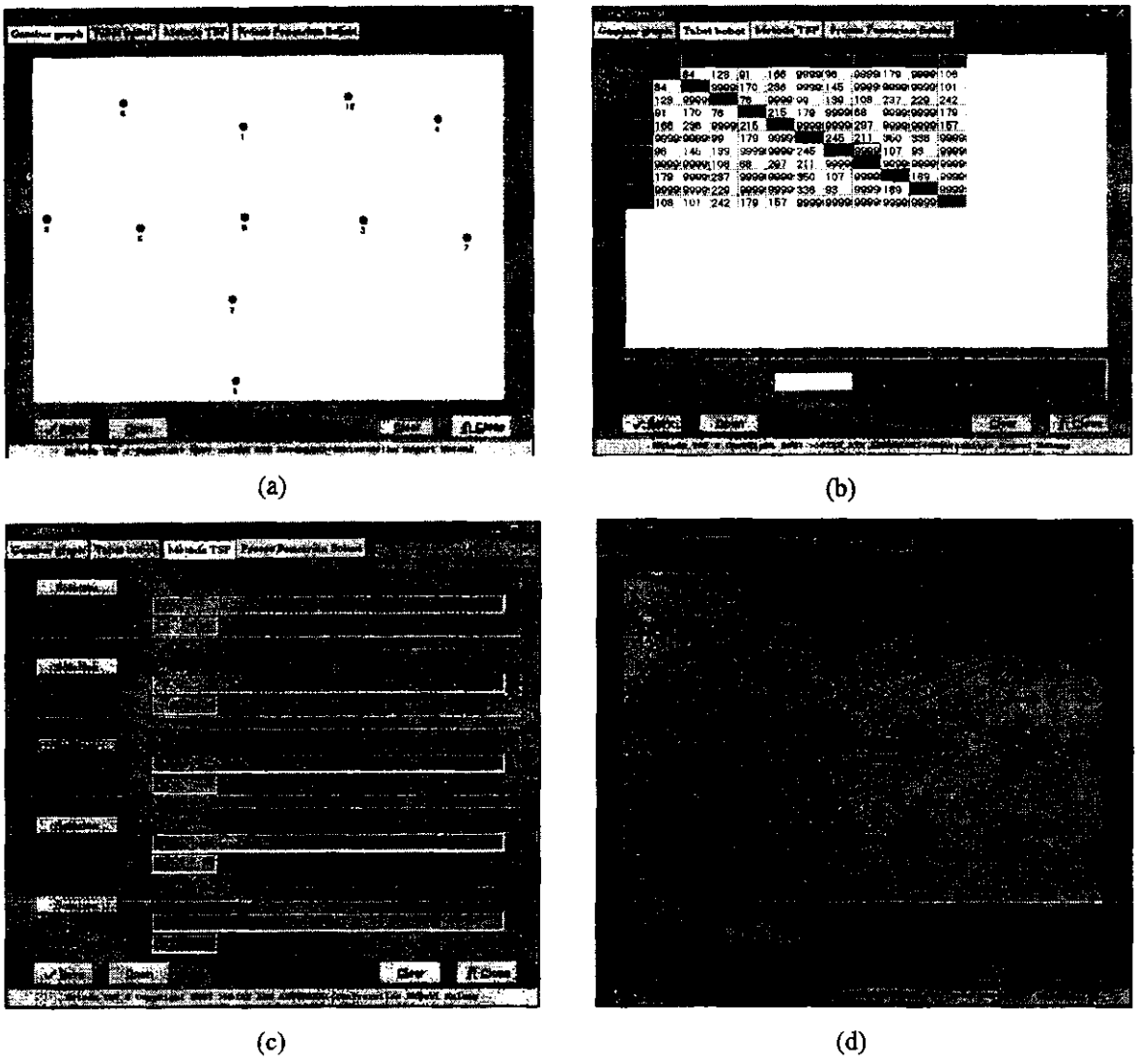
List ini digunakan untuk mencatat sisi yang membentuk lintasan Hamilton (digunakan pada metode *Cheapest Link, Branch dan Bound, Nearest Insertion, Farthest Insertion,* dan *Arbitrary Insertion*). Dengan menggunakan *list,* maka penyisipan sisi bisa dilakukan dengan mudah. Tipe data himpunan untuk membentuk himpunan titik-titik pada Gambar 3 (digunakan pada *Nearest Insertion, Farthest Insertion,* dan *Arbitrary Insertion*). Contoh tampilan eksekusi program dapat dilihat pada Gambar 4.



Gambar 3. *Graph* 11 Titik untuk Permasalahan *TSP*

Tabel 1. Hasil Eksekusi Program Atas Berbagai Metode *TSP*

Metode	Titik awal	Rute yang dihasilkan	Panjang sisi
Nearest Neighbor	A	A–B–K–E–D–H–C–F–G–J–I–A	153,80
Cheapest Link	-	A–B–K–E–H–D–B–F–I–J–G–A	161,00
Nearest Insertion	A	A–I–G–J–C–F–H–D–E–K–B–A	154,30
Farthest Insertion	A	A–B–K–E–D–H–F–C–G–J–I–A	153,50
Arbitrary Insertion	A	I–J–F–C–F–G–D–E–K–A–B–A	144,70
Matriks Bentuk Normal	-	A–B–K–E–D–H–C–F–G–J–I–A	153,80
Branch and Bound	A	A–B–K–E–D–H–F–C–G–J–I–A	153,50



(a) (b)

(c) (d)

Gambar 4. Contoh Tampilan Program. (a) *Entry* Data Titik *Graph*, (b) *Entry* Data Bobot Sisi, (c) Rute dan Jarak yang Dihasilkan, (d) Lintasan yang Dihasilkan

$A =$

	A	B	C	D	E	F	G	H	I	J	K
A	0	8,4	12,3	9,1	16,6	0	9,6	0	17,9	0	10,8
B	8,4	0	0	17,0	23,6	0	14,5	0	0	0	10,1
C	12,3	0	0	7,6	0	9,9	13,9	10,8	23,7	22,9	24,2
D	9,1	17,0	7,6	0	21,5	17,9	0	6,8	0	0	17,9
E	16,6	23,6	0	21,5	0	0	0	29,7	0	0	15,7
F	0	0	9,9	17,9	0	0	24,5	21,1	35,0	33,6	0
G	9,6	14,5	13,9	0	0	24,5	0	0	10,7	9,3	0
H	0	0	10,8	6,8	29,7	21,1	0	0	0	0	0
I	17,9	0	23,7	0	0	35,0	10,7	0	0	18,9	0
J	0	0	22,9	0	0	33,6	9,3	0	18,9	0	0
K	10,8	10,1	24,2	17,9	15,7	0	0	0	0	0	0

Gambar 5. Matriks Bobot untuk Graph pada *Graph* 11 Titik untuk Permasalahan *TSP*

Implementasi *Input* dan *Output* Data

Input titik-titik dalam *graph* dilakukan dengan memanfaatkan mouse. Pengguna cukup mengklik di atas bidang yang disediakan, maka untuk setiap klik akan ditampilkan sebuah titik dengan nomor tertentu. Komponen yang digunakan untuk menerima input adalah *TImage*. Setiap titik yang diinputkan akan dicatat posisinya pada sebuah variabel yang bertipe *TList*. Guna variabel ini adalah untuk keperluan visualisasi *output*.

Bobot diisikan dalam suatu tabel yang ukurannya disesuaikan dengan banyaknya titik yang dimasukkan sebelumnya. Untuk setiap bobot yang diisikan maka akan pula ditampakkan sisi yang menghubungkan dua titik. Tabel pengisian data bobot menggunakan komponen TstringGrid. Untuk setiap bobot yang diisikan, nilainya disimpan dalam array bertipe matriks seperti yang telah dijelaskan. Setelah graph dan bobotnya ditentukan, kemudian sebuah metode dipilih. Hasil output suatu metode akan ditampakkan dalam bentuk teks yang menjelaskan apa yang dilakukan setiap langkah dan visualisasi lintasan Hamilton yang terbentuk. Output teks ditampilkan dalam komponen TMemo, sedangkan visualisasi lintasan Hamilton tetap dalam image yang digunakan untuk input.

Eksekusi Program

Berikut ini akan ditunjukkan hasil eksekusi program. Data yang digunakan untuk mengamati hasilnya adalah pada Gambar 3. Dari permasalahan tersebut,matriks keterhubungannya dapat dilihat pada Gambar 5.

Hasil eksekusi program terhadap data di atas dengan menggunakan metode yang telah disebutkan dapat dilihat pada Tabel 1.

Analisis Kinerja Algoritma

Analisis terhadap setiap algoritma dilakukan dengan membandingkan syarat cukup dari *graph* yang digunakan dan syarat lainnya, serta algoritmanya.

1. Hanya metode matriks bentuk normal dan metode *Branch and Bound* yang tidak mensyaratkan *graph* lengkap, kelima metode lainnya menggunakan *graph* lengkap.
2. Untuk memulai prosesnya, algoritma *Nearest Neighbor, Nearest Insertion, Farthest Insertion, Arbitrary Insertion,* serta *Branch and Bound* mensyaratkan adanya penentuan titik awal. Titik awal yang berbeda memungkinkan menghasilkan lintasan yang berbeda. Metode lainnya tidak memerlukan penentuan titik awal.
3. Algoritma *Nearest Neighbor* dan *Cheapest Link* mungkin menghasilkan sikel yang tidak tunggal karena keduanya menggunakan pemilihan sisi dan ada kemungkinan dijumpai sisi yang bobotnya sama. Metode Matriks Bentuk Normal juga mungkin menghasilkan lebih dari satu sikel karena pemilihan titik yang berbeda pada baris dan kolom awalnya. Algoritma lainnya hanya bisa menghasilkan satu lintasan untuk setiap kali proses.
4. Metode matriks bentuk normal harus bisa mengubah matriks bobot menjadi bentuk normal terlebih dulu untuk mendapatkan lintasan Hamilton, bila tidak bisa maka tidak ada sikel Hamilton yang dihasilkan.
5. Untuk proses penyusunan lintasan, metode *Nearest Insertion, Farthest Insertion, Arbitrary Insertion,* serta *Branch and Bound* menggunakan penyisipan titik, sedangkan metode *Cheapest Link* menggunakan penyisipan sisi.

6. Pada pengujian dengan *graph* komplit dengan dua atau lebih sisi yang memiliki bobot yang sama, metode *Nearest Neighbor, Nearest Insertion Heuristic* dan matriks bentuk normal menghasilkan sikel dengan bobot yang berbeda.
7. Pada pengujian dengan *graph* komplit yang menghasilkan sikel yang tidak tunggal (terdapat dua alternatif rute atau lebih dengan total tiap sisinya memiliki bobot berbeda, metode *NearestNeighbor* dan matriks bentuk normal bobot yang berbeda). Sedangkan untuk algoritma *Cheapest Link* dan *Nearest Insertion Heuristic* diperoleh sikel tunggal dari sembarang titik dengan total bobot yang sama.
8. Pada pengujian dengan *graph* komplit dengan jumlah titik genap, metode *Nearest Insertion Heuristic* menghasilkan sikel dengan bobot yang lebih minimal.

   Pada *graph* komplit dengan jumlah titik ganjil, sikel Hamilton yang dihasilkan dengan menggunakan algoritma matriks bentuk normal, *Branch and Bound*, serta *Nearest Insertion Heuristic* sama, begitu pula dengan bobotnya.

KESIMPULAN

Dalam teori *graph*, *TSP* adalah masalah pencarian jarak dari lintasan (*path*) terpendek yang melewati setiap titik tepat sekali dan kembali ke titikawal kembali. Masalah ini banyak ditemui dalam kegiatan pendistribusian barang, di mana salesman harus bisa menuju semua tujuan dengan jarak terpendek.

Ada banyak metode TSP, diantaranya adalah metode *Nearest Neighbor, Cheapest Link, Nearest Insertion, Farthest Insertion, Arbitrary Insertion, matriks bentuk normal,* serta *Branch and Bound.*Masing-masing metode memiliki karakteristik tersendiri. Untuk mendapatkan hasil dengan akurat dan cepat, maka dibuatlah sebuah program dengan bahasa *Delphi.*

Dari hasil percobaan dengan *graph* tidak lengkap 11 titik, program menunjukkan bahwa metode *Farthest Insertion*dan *Branch and Bound*bisa menghasilkan rute terpendek. Akan tetapi kesimpulan ini masih perlu diuji lain dengan data yang lebih terstandar.

DAFTAR RUJUKAN

Andriani, Y.A. 2005. *PenggunaanMetode Nearest Neighbor dan Metode Cheapest Link dalam Penyelesaian Traveling Salesman Problem.* Skripsi tidak diterbitkan. Malang: FMIPA Universitas Negeri Malang.

Applegate, D., Cook, W. & Rohe, A. 2003. Chained Lin-Ker-Nighan for Large Traveling Salesman Problems.*INFORMS Journal on Computing*, 15(1): 82–92.

Bentley, J.L.1990.*Experiments on Traveling Salesman Heuristics*. Proceedings of The First Annual ACM-SIAM Symposium on Discrete Algorithms, 91.

Christofides, N. 1976.*Worst-Case Analysis of A New Heuristic for The Traveling Salesman Problem*. Tech. Report 388, Carnegie Mellon University, Pittsburgh, USA.

Croes, G.A. 1958. A Method for Solving Traveling Salesman Problem. *OperationsResearch,* 6(6): 791–812.

Current, J.R. & Schilling, D.A. 1994.The MedianTour and Maximal Covering Tour Problems: Formulationsand Heuristics; *Eur Journal OperationResearch*, 73: 114–126.

Dantzig, G., Fulkerson,D. & Johnson, S. 1954. Solution of a Large-scale Traveling Salesman Problem.*Operations Research*, 2:393–410.

Freisleben, B. & Merz, P.1996. *A Genetic Local Search Algorithm for Solving Symmetric and ASymmetric Traveling SalesmanProblems.* Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, Nagoya,Japan, 616-621.

Harary, F. 1983. *Graph Theory.* California: Addison-Wesley Publishing Company.

Hasnawati, I.Q. 2005. *Penggunaan Metode Matriks Bentuk Normal dan Metode Branch and Bound untukMenyelesaikan Persoalan Traveling Salesman Problem.* Skripsi tidak diterbitkan. Malang: FMIPA Universitas Negeri Malang.

Herdhiyanto, F. 2005. *Penyelesaian Masalah Travelling Salesman Problem Menggunakan Algoritma Koloni Semut.* Skripsi tidak diterbitkan. Malang: FMIPA Universitas Negeri Malang.

Lawer, E.L. 1987. *The Traveling Salesman Problem; AGuided Tour of Combinatorial Optimization.* New York: John Wiley & Sons.

Nurhidayah, Y. 2005. *Penyelesaian Travelling Salesman Problem dengan Menggunakan Metode Nearest Insertion Heuristic.* Skripsi tidak diterbitkan. Malang: FMIPA Universitas Negeri Malang.

Rosenkrantz, D.J., Stearns & Lewis, P.M. 1977. An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM Journal of Computation*, 6:563-581.

Wilson, R.J. & Watkins, J.J. 1990. *Graph An Introductory Approach.* Canada: John Wiley and Sons, Inc.