Penentuan Rute Perjalanan *Multi Travelling Salesman Problem (m-TSP)* pada Mobil Patroli Keamanan Polisi dengan Metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A*

TUGAS AKHIR

Oleh:
SANTI DWI NURHUMAM
0310960071-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MIPA
UNIVERSITAS BRAWIJAYA
MALANG

2007

Penentuan Rute Perjalanan *Multi Travelling Salesman Problem (m-TSP)* pada Mobil Patroli Keamanan Polisi dengan Metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A*

TUGAS AKHIR

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

Oleh:
SANTI DWI NURHUMAM
0310960071-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MIPA
UNIVERSITAS BRAWIJAYA
MALANG
2007


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN TUGAS AKHIR

PENENTUAN RUTE PERJALANAN *MULTI TRAVELLING SALESMAN PROBLEM (m-TSP)* PADA MOBIL PATROLI KEAMANAN POLISI DENGAN METODE *HEURISTIC ASSIGNMENT* FISHER-JAIKUMAR DAN ALGORITMA A*

Oleh:
SANTI DWI NURHUMAM
0310960071-96

Setelah dipertahankan di depan Majelis Penguji
Pada tanggal 31 Juli 2007
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

Pembimbing	**Mengetahui,** **Ketua Jurusan Matematika** **Fakultas MIPA Universitas** **Brawijaya**
Wayan Firdaus M, S.Si, MT	**Dr. Agus Suryanto, MSc.**
NIP. 132 158 724	**NIP. 132 126 049**


UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama : Santi Dwi Nurhumam
NIM : 0310960071
Jurusan : Matematika
Penulis tugas Akhir berjudul : Penentuan Rute Perjalanan *Multi Travelling Salesman Problem (m-TSP)* pada Mobil Patroli Keamanan Polisi dengan Metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A*

Dengan ini menyatakan bahwa :

1. **Isi dari tugas Akhir yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam Tugas Akhir ini.**
2. **Apabila dikemudian hari ternyata Tugas Akhir yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 31 Juli 2007
Yang menyatakan,

(Santi Dwi Nurhumam)
NIM. 0310960071


UNIVERSITAS BRAWIJAYA

Penentuan Rute Perjalanan *Multi Travelling Salesman Problem (m-TSP)* pada Mobil Patroli Keamanan Polisi dengan Metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A*

ABSTRAK

Perjalanan n mobil patroli (di mana n lebih dari satu) ke sejumlah m titik rawan (di mana m lebih dari n) dengan asumsi titik (1) sebagai kantor polisi (tempat keberangkatan semua mobil patroli) termasuk dalam *m-TSP* (*Multi Travelling Salesman Problem*). Dalam tugas akhir ini akan dibangun sebuah perangkat lunak yang mampu menyelesaikan permasalahan tersebut dengan mengelompokkan terlebih dahulu m-1 (kecuali titik rawan (1)) titik rawan ke dalam n sub tur, di mana satu titk rawan hanya boleh menjadi anggota sebuah sub tur. Pengelompokkan ini menggunakan metode *heuristic assignment* Fisher-Jaikumar. Selanjutnya masing-masing sub tur direpresentasikan dalam bentuk graf, lalu dilakukan optimasi pencarian rute berdasarkan jarak pada masing-masing sub tur menggunakan algoritma A*, hasilnya berupa rute yang harus ditempuh oleh masing-masing mobil patroli. Untuk mengevaluasi hasil dari metode yang digunakan, dilakukan perbandingan dengan hasil perhitungan pada semua kemungkinan. Perbandingan dilakukan pada lima kali uji coba menggunakan tujuh, delapan, sembilan, sepuluh dan sebelas titik rawan. Pada perhitungan jarak menggunakan metode *heuristic assignment* Fisher-Jaikumar dan algoritma A* menghasilkan galat rata-rata sebesar 20,43% dari jarak minimum sebenarnya. Sedangkan waktu pemrosesannya jauh lebih cepat untuk jumlah titik rawan lebih dari tujuh daripada dengan menghitung semua kemungkinan.

Multi Travelling Salesman Problem (m-TSP) Journey Route Determination in The Police Patrol Cars with Fisher-Jaikumar Heuristic Assignment Methods and A* Algorithm

ABSTRACT

The journey of n patrol cars (more than one car) appoints to sensitive points m (more than n) assuming that point (1) refers to a police office (where all patrol cars depart from) included within m-TSP (Multi Travelling Salesman Problem). This final assignment concerns with a software to solve this problem through initially grouping m-1 (except sensitive point (1)) into sub-tour n, where only one sensitive point registers into a sub-tour membership. The grouping considers Fisher-Jaikumar *heuristic assignment* method. Graphic subsequently represents each sub-tour continued with optimization of route tracking based on each sub-tour distance using A* algorithm, resulting in a route has to be taken by each patrol car. Evaluating the result of method application requires comparing measurement result of any possibilities. The comparison involves five times of trial covering seven, eight, nine, ten and eleven sensitive points. Distance measurement also embarks on Fisher-Jaikumar *heuristic assignment* methods and A* algorithm to produce average deviation 20.43% of actual minimum distance. Time to process seems more faster for more than seven sensitive points rather than counting any possibilities.


UNIVERSITAS BRAWIJAYA

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Allah SWT karena hanya dengan berkat rahmat, taufiq dan hidayah-Nya pada akhirnya penulis dapat menyelesaikan tugas akhir yang menjadi syarat kelulusan program Strata I. Shalawat serta salam tercurah kepada Nabi Besar Muhammad SAW beserta seluruh keluarga, sahabat dan pengikutnya.

Selesainya tugas akhir ini tidak lepas dari dukungan berbagi pihak. Oleh karena itu penulis dengan rendah hati menghaturkan terima kasih yang tak terhingga kepada:

1. Bapak Wayan Firdaus Mahmudy, S.Si, MT selaku pembimbing dan Ketua Program Studi Ilmu Komputer Universitas Brawijaya Malang, yang telah memberikan saran, kritik serta bimbingannya selama ini kepada penulis.
2. Ibu Dian Eka Ratnawati, S.Si, M.Kom selaku penasehat akademik yang telah memberikan dukungannya.
3. Bapak dan Ibu, terima kasih atas doa-doa yang selalu mengiringi perjalanan penulis.
4. Mas Hemmy, mbak Lia serta calon keponakan penulis atas semangat yang selama ini diberikan pada penulis.
5. Sahabat terbaik dalam hidup penulis, WeDe terima kasih atas inspirasinya.
6. Sahabat seperjuangan penulis, Ratna, Ruli, Nana, Yani dan Amarisa terima kasih atas semangat yang tidak pernah berhenti diberikan.
7. Teman-teman kos penulis Vivi, Ratri dan Diah atas bantuan dan doa yang diberikan pada penulis.
8. Sahabat dan teman-teman penulis atas segala dukungan, semangat dan bantuannya selama ini.
9. Seluruh rekan-rekan Ilmu Komputer angkatan 2003 atas bantuan yang telah diberikan.

Terima kasih juga penulis persembahkan kepada pihak-pihak yang belum disebutkan satu per satu yang pernah berjasa dalam memberikan bantuan dan semangat. Semoga apa yang penulis persembahkan dapat memberi manfaat bagi yang membutuhkan.

Tugas akhir ini masih jauh dari kesempurnaan, untuk itu penulis mengharapkan saran dan kritik yang membangun dari para pembaca.

Malang, 31 Juli 2007

Penulis

DAFTAR GAMBAR

DAFTAR TABEL


UNIVERSITAS BRAWIJAYA

DAFTAR LAMPIRAN


UNIVERSITAS BRAWIJAYA

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Latar Belakang

Masalah pendistribusian barang dari satu agen ke tempat tujuan yang tersebar di berbagai tempat sering ditemui dalam kehidupan sehari-hari. Misalnya saja perjalanan suatu mobil boks yang harus mengantarkan barang-barang pesanan ke toko-toko kecil atau perjalanan mobil travel untuk menjemput para penumpangnya. Jika dalam kasus tersebut hanya digunakan sebuah mobil boks atau sebuah mobil travel (dalam *TSP* dikenal dengan digunakannya seorang *salesman*), pada sekali keberangkatan, maka permasalahan tersebut dikenal sebagai *Travelling Salesman Problem (TSP)*.

Namun untuk permasalahan di mana jumlah *salesman* yang digunakan lebih dari satu disebut sebagai *multi Traveling Salesman Problem (m-TSP)*. Misalnya saja kasus penjemputan penumpang mobil travel. Di mana untuk sekali keberangkatan perusahaan travel menggunakan jumlah mobil lebih dari satu. Permasalahan yang dihadapi pada kasus *m-TSP* ini sama dengan *TSP* yaitu bagaimana meminimumkan jarak yang ditempuh, namun dalam penyelesaiannya terdapat perbedaan.

Beberapa penulis telah melakukan penelitian menganai *m-TSP* antara lain Bellmore 1974, Berrenguer 1978, Desrois 1988, Gavish 1976, Jonker 1988, Lawler 1985, Lenstra 1979 dan Orloff 1974. Namun penelitian tersebut hanya terbatas pada *m-TSP* yang dapat direpresentasikan dengan graf lengkap (Budi, 2002:56). Dalam tugas akhir ini akan dibahas penyelesaian kasus *m-TSP* menggunakan metode *heuristic assignment* dan algoritma A*.

Penggunaan metode *heuristic assignment* selain untuk menjamin semua titik akan dikunjungi, juga untuk menjamin tidak adanya titik yang dikunjungi lebih dari sekali. Karena metode ini membagi n kota ke dalam m sub tur. Sehingga tidak ada kota yang tidak masuk dalam sub tur.

Algoritma A* dipilih karena algoritma ini bekerja dari sebuah perkiraan terbaik biaya yang dimiliki jalur-jalur yang melewati sebuah kota, sehingga biaya dari kota tersebut menuju ke kota tujuan adalah biaya yang paling minimum. Dengan kata lain ketika algoritma A* mengakhiri pencariannya, berarti algoritma tersebut telah menemukan jalur yang benar-benar memiliki nilai minimum.

Algoritma A* tidak akan pernah mengabaikan kemungkinan adanya jalur lain yang memiliki biaya lebih rendah. Contoh algoritma *TSP* yang lain yaitu *Depth-First Search, Breadth-First Search, Djikstra.*

Salah satu contoh kasus *m-TSP* yang akan dibahas dalam tugas akhir ini adalah perjalanan sejumlah m mobil patroli dari kantor menuju ke n daerah rawan kemudian kembali lagi ke kantor. Untuk nilai m dan n lebih dari satu. Permasalahan tersebut meliputi *m-TSP* yang direpresentasikan dengan graf lengkap serta *m-TSP* yang direpresentasikan dengan graf tidak lengkap. Oleh karena itu mungkin saja dalam suatu graf tidak memuat lintasan yang mengunjungi semua titik dalam graf tersebut (sirkuit Hamilton) yang merupakan permasalahan *Graphical Travelling Salesman Problem (GTSP),* yang penyelesainnya memungkinkan sebuah daerah rawan dapat dikunjungi lebih dari satu kali (Fonlupt 1993, Heru 2000, Budi 2002:56). Kasus tersebut merupakan pengecualian di mana daerah rawan bisa dikunjungi lebih dari satu kali karena adanya *GTSP* dalam *m-TSP*. dengan penggabungan metode *heuristic assignment* dan algoritma A*, diharapkan bisa menghasilkan solusi yang mendekati optimum.

Dari seluruh uraian tersebut maka penulis mengambil judul tugas akhir **"Penentuan Rute Perjalanan *Multi Travelling Salesman Problem (m-TSP)* pada Mobil Patroli Keamanan Polisi dengan Metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A*".**

1.2 Perumusan Masalah

Berdasarkan uraian pada latar belakang di atas maka dalam tugas akhir ini dapat dirumuskan perumusan masalah yaitu bagaimana hasil penerapan metode *heuristic assignment* Fisher-Jaikumar dan algoritma A* pada kasus perjalanan mobil patroli dengan mobil lebih dari satu?

1.3 Batasan Masalah

Dari permasalahan di atas, berikut ini diberikan batasan masalah untuk menghindari melebarnya masalah yang akan diselesaikan:

1. Untuk menghindari suatu daerah rawan dikunjungi lebih dari sebuah mobil patroli, maka jumlah mobil patroli dibatasi paling banyak sejumlah daerah rawan kurang satu.

2. Tiap mobil patroli mempunyai bobot yang sama. Dengan kata lain tiap mobil mempunyai hak yang sama untuk mengunjungi suatu daerah rawan.
3. Dalam tugas akhir ini penentuan rute perjalanan hanya berdasarkan jarak.
4. Jumlah mobil yang digunakan minimal dua.

1.4 Tujuan Penelitian

Tujuan yang ingin dicapai dari penelitian ini adalah mengetahui tingkat keoptimalan rute yang dihasilkan oleh perangkat lunak menggunakan metode *heuristic assignment* Fisher-Jaikumar dan algoritma A*.

1.5 Manfaat Penelitian

Menyediakan suatu aplikasi yang mampu menyelesaikan *m-TSP* dalam menentukan rute perjalanan khususnya permasalahan pada mobil patroli keamanan polisi.

BAB II
TINJAUAN PUSTAKA

2.1 Transportasi

Transportasi merupakan salah satu permasalahan yang sering kita hadapi dalam kehidupan sehari-hari. Salah satu contoh permasalahan yang timbul dalam bidang ini yaitu rute manakah yang memiliki biaya paling murah untuk dilalui seorang *salesman* yang harus mengunjungi sejumlah daerah, di mana tiap daerah harus dikunjungi tepat satu kali kemudian kembali lagi ke tempat semula. Permasalahan tersebut disebut sebagai *Travelling Salesman Problem (TSP)*. Sedangkan untuk permasalahan di mana terdapat lebih dari seorang *salesman* disebut sebagai *multi Travelling Salesman Problem (m-TSP)*.

Permasalahan *TSP* maupun *m-TSP* dimodelkan dalam bentuk graf lengkap. Pada kasus *m-TSP* graf lengkap tersebut memiliki n buah simpul yang menyatakan daerah-daerah yang harus dikunjungi oleh sejumlah m *salesman*. Sedangkan bobot pada tiap garis pada graf tersebut menyatakan jarak dari tiap daerah yang harus dikunjungi oleh *salesman*.

Secara matematis *m-TSP* bisa diformulasikan dalam Persamaan 2.1 (Gavish 1986, Kulkarni 1985, Budi 2002:57):

$$Z = \min\left\{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}\right\} \quad \text{dengan kendala} \tag{2.1}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \text{ untuk j = 1, 2, 3, ..., } n\text{-1} \tag{2.2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \text{ untuk i = 1, 2, 3, ..., } n\text{-1} \tag{2.3}$$

$$\sum_{i=1}^{n} x_{i1} = m \tag{2.4}$$

$$\sum_{j=1}^{n} x_{1j} = m \qquad (2.5)$$

$x_{ij} = 1$, apabila ada perjalanan *salesman* dari simpul i menuju simpul j

$x_{ij} = 0$, apabila tidak ada perjalanan *salesman* dari simpul i menuju simpul j

c_{ij}, menyatakan jarak dari simpul i menuju simpul j.

Persamaan (2.2) dan Persamaan(2.3) menjamin bahwa setiap simpul hanya dikunjungi sekali oleh *salesman.* Sedangkan Persamaan (2.4) dan Persamaan (2.5) menjamin bahwa sejumlah m *salesman* melakukan tur. Untuk kasus dalam tugas akhir ini, yaitu menentukan rute perjalanan mobil patroli keamanan polisi, simpul merupakan representasi dari kantor polisi dan pusat kerawanan yang harus dikunjungi sedangkan *salesman* merupakan representasi dari mobil patroli yang digunakan itu sendiri.

2.1.1 Graf

Sebuah graf G = (V,E) terdiri dari sebuah himpunan titik-titik V yang tidak boleh kosong serta himpunan garis-garis E (*edge*). Jika garis-garis tersebut diasumsikan sebagai pasangan berurutan dari titik-titik (v,w) maka disebut graf berarah; v disebut sebagai ekor dan w sebagai kepala dari garis (v,w). Jika garis-garis tersebut merupakan pasangan titik-titik yang tidak berurutan, tetap dilambangkan sebagai garis (v,w), namun disebut sebagai graf tidak berarah (Aho et al, 1974:50). Titik-titik tersebut biasa disebut sebagai *node* atau *vertex*. Sedangkan jumlah garis (*edge*) yang menghubungkan suatu *node* dengan *node* lain disebut derajat.

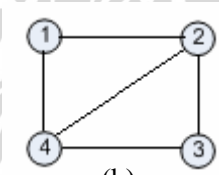
(a) (b)

Gambar 2.1 (a) Graf Tidak Berarah; (b) Graf Berarah.

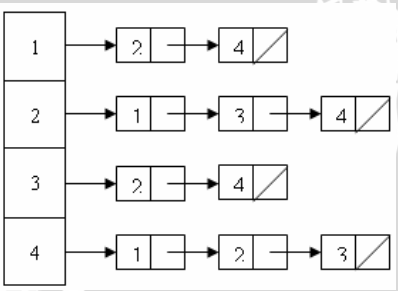
Gambar 2.1 (a) merupakan graf tidak berarah yang memiliki empat *node* yaitu *node* 1, 2, 3 dan 4. *Node* 1 memiliki derajat 2, *node* 2 memiliki derajat 3, *node* 3 memiliki derajat 2 dan *node* 4 memiliki derajat 4. Sedangkan Gambar 2.1 (b) merupakan graf berarah yang memiliki empat buah *node* yaitu *node* 1, 2, 3 dan 4. untuk jumlah derajat pada masing-masing *node* sama dengan Gambar 2.1 (a).

Terdapat beberapa representasi dari graf antara lain berupa himpunan yang digunakan dalam bidang matematika, dalam bentuk gambar digunakan untuk lebih memperjelas serta berupa *adjacency list* dan *adjacency matrix* yang biasa digunakan dalam bidang komputer. *Adjacency list* berupa daftar titik yang berhubungan dengan suatu titik lain sedangkan *adjacency matrix* berupa elemen *matrix* [i][j] yang berisi satu nilai tertentu jika titik-titik tersebut berhubungan.

V = {1,2,3,4}
E = {{1,2},{1,4},{2,3},{2,4},{3,4}}
(a.)



(b)

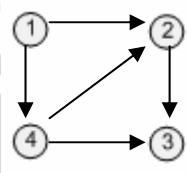


(c)

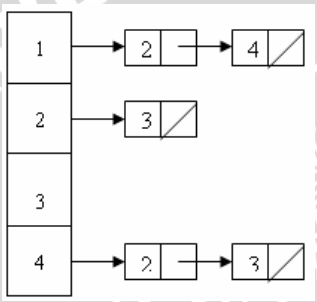
$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$
(d)

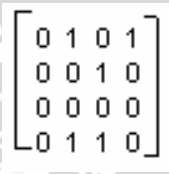
Gambar 2.2 Representasi Graf Tidak Berarah: (a) Himpunan; (b) *Pictorial Representation*; (c) *Adjacency List*; (d) *Adjacency Matrix*.

V = {1,2,3,4}
E = {{1,2},{1,4},{2,3},{4,2},{4,3}}



(a) (b)





(c) (d)

Gambar 2.3 Representasi Graf Berarah: (a) Himpunan; (b) *Pictorial Representation*; (c) *Adjacency List*; (d) *Adjacency Matrix*.

2.1.2 Graf Hamilton

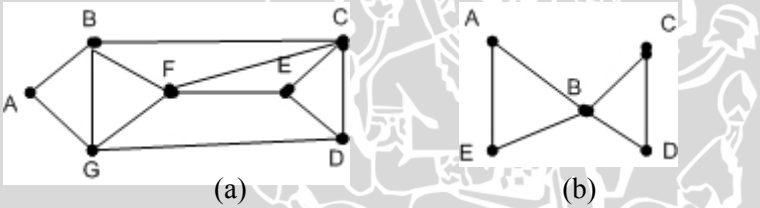
Suatu graf disebut sebagai graf Hamilton apabila terdapat sirkuit dalam graf tersebut yang mengunjungi tiap titik tepat satu kali, kecuali titik awal yang sama dengan titik akhirnya. Apabila graf tersebut merupakan graf dengan lintasan terbuka (titik awalnya berbeda dengan titik akhirnya) maka disebut sebagai graf semi Hamilton.

Karakteristik graf Hamilton dijelaskan dalam teorema Bondy-Chvatal pada tahun 1972 berbunyi, ”Sebuah graf disebut Hamilton jika dan hanya jika graf tersebut tertutup”. Dengan kata lain semua graf yang tertutup dengan sempurna merupakan graf Hamilton. Pernyataan Bondy-Chvatal tersebut mengacu pada teorema yang sudah ada sebelumnya, yaitu teorema yang diungkapkan oleh Dirac

(1952) yang berbunyi, ”Sebuah graf sederhana yang memiliki *vertex* sejumlah n (n>2) disebut sebagai graf Hamilton jika masing-masing *vertex* memiliki derajat n/2 atau lebih besar” serta teorema yang diungkapkan oleh Ore (1960) yang berbunyi, ”Sebuah graf dengan *vertex* sejumlah n (n>2) disebut sebagai graf Hamilton jika jumlah derajat dari dua *vertex* yang tidak berhubungan adalah n atau lebih besar dari n”.Untuk menentukan sebuah graf termasuk graf Hamilton atau bukan kita bisa menggunakan suatu petunjuk.

Jika G merupakan graf Hamilton maka G mempunyai subgraf H dengan sifat-sifat sebagai berikut (Jong, 2002: 222):

1. H terhubung
2. H memuat semua titik G
3. H mempunyai jumlah garis yang sama dengan jumlah titiknya
4. setiap titik dalam H mempunya derajat 2.



(a) (b)

Gambar 2.4 (a) Graf Hamilton; (b) Bukan Graf Hamilton

Misalkan graf pada Gambar 2.4 (a) adalah graf G, di dalamnya terdapat subgraf H yaitu ABCDEFGA. Subgraf tersebut telah memuat semua titik yang ada dalam graf G. Jumlah titiknya adalah tujuh dan jumlah garisnya adalah tujuh. Dengan kata lain jumlah titik dan jumlah garis dalam subgraf H adalah sama. Dalam subgraf H semua titik mempunyai derajat dua. Sehingga dapat disimpulkan bahwa graf G adalah graf Hamilton.

Misalkan Gambar 2.4 (b) adalah graf G. Kita bisa membentuk subgraf H yang memuat semua titiknya yaitu ABDCBEA atau AEBCDBA ataupun yang lain, namun tetap saja B berderajat empat. Sehingga subgraf H tidak memenuhi syarat yang keempat.

Dalam suatu kasus yang membutuhkan graf Hamilton, mungkin saja terjadi masukan yang ada berupa graf terbuka yang berarti bukan

merupakan graf Hamilton. Permasalahan seperti ini termasuk dalam *Graphical Travelling Salesman Problem (GTSP)*. Permasalahan ini bisa diselesaikan dengan cara menambahkan busur pada pasangan simpul yang belum bertetangga.

2.2 *Generalized Assignment Problem (GAP)*

Untuk menyelesaikan *m-TSP* mula-mula harus dilakukan pembagian sejumlah titik rawan kepada sejumlah *salesman.* Permasalahan tersebut termasuk dalam *Generalized Assignment Problem (GAP)*. Total biaya yang dikenakan bisa saja berubah tergantung pada titik mana saja yang harus dikunjungi oleh seorang *salesman* (*agen-task assignment*). Dalam GAP *salesman* direpresentasikan sebagai agen dan titik rawan sebagai pekerjaan yang harus dikerjakan oleh agen. Dalam permasalahan ini dibutuhkan tepat seorang agen untuk menyelesaikan masing-masing pekerjaan sehingga bisa diperoleh total biaya yang minimum. Dengan kata lain jika diberikan himpunan N yang berisi sejumlah n pekerjaan dan himpunan M yang berisi sejumlah m agen (tiap pekerjaan harus diselesaikan oleh tepat seorang agen). Formula dari permasalahan tersebut ditunjukkan pada Persamaan 2.6.

$$\max \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \qquad \text{dengan kendala} \tag{2.6}$$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad i = 1..., n \tag{2.7}$$

$$\sum_{i=1}^{n} a_{ij}x_{ij} \le b_j \qquad j = 1..., m \tag{2.8}$$

$$x_{ij} \in \{0,1\} \qquad i = 1..., n, \quad j = 1...,m \tag{2.9}$$

Di mana $_i \in N$ dan $_j \in M$, x_{ij} adalah variabel yang berarti apakah pekerjaan $_i$ dikerjakan oleh agen $_j$ atau tidak, c_{ij} adalah biaya yang dibutuhkan agen $_j$ untuk mengerjakan pekerjaan $_i$, a_{ij} adalah jumlah sumber (*resource*) yang dibutuhkan untuk oleh pekerjaan $_i$ ketika

diselesaikan oleh agen $_j$, sedangkan b_j adalah jumlah sumber yang dimiliki oleh agen $_j$ (Alfandari et al, 2001).

Salah satu metode yang bisa digunakan untuk menyelesaikan GAP adalah metode heuristik yang digunakan pada algoritma Fisher-Jaikumar.

2.2.1 Algoritma *Heuristic Assignment* Fisher-Jaikumar

Heuristik merupakan salah satu teknik yang digunakan untuk mempercepat pencarian solusi. Teknik heuristik digunakan untuk mengeliminasi beberapa kemungkinan solusi tanpa harus mengeksplorasinya secara penuh. Selain itu, teknik heuristik juga membantu memutuskan kemungkinan solusi mana yang pertama kali perlu dievaluasi (Munir, 2004).

Heuristik merupakan suatu seni dan ilmu menemukan. Kata heuristik sendiri berasal dari bahasa Yunani yaitu *eureka* yang berarti menemukan. Kata *heureka* yang berasal dari *eureka* pernah dilontarkan oleh seorang matematikawan Yunani yaitu Archimedes, yang berarti ”Saya telah menemukannya”.Suatu metode heuristik yang bagus akan mampu mengurangi waktu yang dibutuhkan untuk memecahkan suatu masalah. Karena dalam metode ini akan dilakukan pengeliminasian kemungkinan solusi yang tidak perlu.

Algoritma Fisher-Jaikumar adalah sebuah algoritma heuristik yang dikembangkan oleh Fisher dan Jaikumar pada tahun 1981. Berikut ini langkah-langkah untuk menyelesaikan kasus *m-TSP* dengan algoritma Fisher-Jaikumar:

1. Diberikan masukan berupa matrik jarak dari tiap-tiap kota serta jumlah *salesman*.
2. Kemudian dipilih salah satu kota sebagai titik awal (*start node*).
3. Lalu dilakukan pengecekan apakah jumlah kota kurang dari atau sama dengan jumlah *salesman* kurang satu, jika tidak, jumlah *salesman* diubah menjadi sama dengan jumlah kota kurang satu, kemudian dilakukan pemilihan pusat *cluster* yaitu sesuai dengan jumlah kota kurang satu. Jika jumlah kota sudah kurang dari atau sama dengan jumlah *salesman*, maka langsung dilakukan pemilihan pusat *cluster*.
4. Berikutnya pemilihan pusat *cluster* dengan cara memilih simpul tetangga *start node* mulai dari jarak terkecil sebanyak jumlah *salesman*.
5. Kemudian diperoleh pusat *cluster* sebanyak jumlah *salesman*.

6. Selanjutnya untuk setiap titik rawan yang belum dikunjungi (bukan termasuk *start node* maupun pusat *cluster*), kota tersebut ditentukan untuk masuk dalam sub tur (*cluster*) yang mana, dengan cara dicari pusat *cluster* yang bertetangga paling dekat dengan kota tersebut. Jika terdapat lebih dari satu pusat *cluster* memiliki jarak terkecil yang sama maka prioritaskan pada *cluster* dengan jumlah titik rawan terkecil. Ulangi langkah tersebut sampai semua kota telah dikunjungi.
7. Kemudian dihasilkan sub tur sebanyak jumlah *salesman*.
8. Algoritma ini memberikan *output* berupa matrik jarak kota pada masing-masing sub tur.

Untuk lebih jelasnya, berikut ini diberikan contoh penggunaan algoritma Fisher-Jaikumar. Misal diberikan 10 *node* dengan matrik jarak seperti dalam Tabel 2.1 dan tiga orang *salesman* sebagai masukan. Pada kasus ini diasumsikan jalur antar *node* merupakan jalan dengan dua arah sehingga direpresentasikan dengan graf tak berarah. Berarti jika diberikan *node* (1) dan *node (*2), maka jarak *node* (1) ke *node* (2) sama dengan jarak *node* (2) ke *node* (1). Pada kenyataannya nanti jarak *node* (1) ke *node* (2) belum tentu sama dengan jarak *node* (2) ke *node* (1). Nilai 0 pada matrik jarak berarti tidak ada jalur (*path*) yang menghubungkan antara dua kota tersebut. Dan *node* (1) diasumsikan sebagai *start node*.

Tabel 2.1 Matrik Jarak

Node ke	1	2	3	4	5	6	7	8	9	10
1	0	9	7	15	0	4	9	0	11	15
2	9	0	2	0	8	10	12	3	0	0
3	7	2	0	7	0	6	5	4	3	9
4	15	0	7	0	15	4	0	8	1	13
5	0	8	0	15	0	6	6	7	6	0
6	4	10	6	4	6	0	8	0	5	4
7	9	12	5	0	6	8	0	4	0	5
8	0	3	4	8	7	0	4	0	15	6
9	11	0	3	1	6	5	8	15	0	7
10	15	0	9	13	0	4	5	6	7	0

Kemudian diberi masukan berupa jumlah *salesman* yang digunakan.

Jumlah *salesman* = 3

Berdasarkan masukan dapat diketahui bahwa

jumlah *salesman* < jumlah kota, sehingga
jumlah sub tur = jumlah *salesman* = 3

Selanjutnya dibuat sub tur. Mula-mula dilakukan pencarian tiga kota yang mempunyai jarak paling dekat dengan *node* (1) (*start node*).

Start node = *node* 1
Sub tur 1 = *node* 6
Sub tur 2 = *node* 3
Sub tur 3 = *node* 2

Node-node yang telah dihasilkan tersebut kemudian dijadikan pusat *cluster* masing-masing sub tur. Yaitu *node* (6) merupakan pusat *cluster* sub tur 1, *node* (3) merupakan pusat *cluster* sub tur 2 dan *node* (2) merupakan pusat *cluster* dari sub tur 3. Sedangkan untuk *node-node* yang belum terpilih yaitu *node* 4, 5, 7, 8, 9 dan 10, dicari jarak paling dekat dengan pusat *cluster* yang mana, lalu dimasukkan dalam anggota sub tur tersebut.

Anggota sub tur 1 = *node* 4, 5, 6, 10
Anggota sub tur 2 = *node* 3, 7, 9
Anggota sub tur 3 = *node* 2, 8

Dari metode *heuristic assignment* Fisher-Jaikumar tersebut dihasilkan matrik jarak masing-masing sub tur seperti ditunjukkan pada Tabel 2.2, Tabel 2.3 dan Tabel 2.4.

Tabel 2.2 Matrik Jarak Sub Tur 1

Node ke	1	4	5	6	10
1	0	15	0	4	15
4	15	0	15	4	13
5	0	15	0	6	0
6	4	4	6	0	4
10	15	13	0	4	0

Tabel 2.3 Matrik Jarak Sub Tur 2

Node ke	1	3	7	9
1	0	7	9	11
3	7	0	5	3
7	9	5	0	8
9	11	3	8	0

Tabel 2.4 Matrik Jarak Sub Tur 3

Node ke	1	2	8
1	0	9	0
2	9	0	3
8	0	3	0

2.3 Algoritma A*

Algoritma A* pertama kali diperkenalkan pada tahun 1968 oleh Peter Hart, Nils Nilsson dan Bertram Raphael. Dalam tulisan mereka, algoritma ini disebut sebagai algoritma A.

Algoritma A merupakan salah satu algoritma pencarian graf, yaitu menemukan jalur terpendek dari titik awal sampai ke titik tujuan dalam suatu graf. Algoritma ini menggunakan perkiraan *heuristic* h(x) yang menyusun tiap titik x berdasarkan perkiraan rute terbaik yang melalui titik tersebut. Kemudian titik yang dikunjungi adalah titik dengan perkiraan terbaik tersebut.

Algoritma A bekerja dengan menyimpan sebuah antrian prioritas dari garis-garis yang melalui graf tersebut dimulai dari titik awal. Misalkan dari titik awal (*start node*) ke titik A terdapat sebuah garis, prioritas pada garis x berdasarkan Persamaan 2.10.

$$f(x) = g(x) + h(x) \tag{2.10}$$

Pada Persamaan 2.10, g(x) merupakan biaya yang dimiliki oleh garis yang menghubungkan dari titik awal menuju ke titik x. Sedangkan h(x) adalah biaya minimum yang sebenarnya untuk mencapai titik tujuan dari titik x. Nilai f(x) yang paling rendah mendapat prioritas paling tinggi dalam antrian tersebut.

Algoritma A ini kemudian dikembangkan sehingga menjadi algoritma A*. Jika dalam algoritma A digunakan nilai h(x) maka dalam algoritma A* digunakan nilai h'(x) yang merupakan perkiraan biaya untuk sampai pada suatu tujuan dari titik x. Dengan syarat $h'(n) \leq h(x)$. Untuk evaluasinya algoritma ini menggunakan fungsi seperti yang terlihat pada Persamaan 2.11.

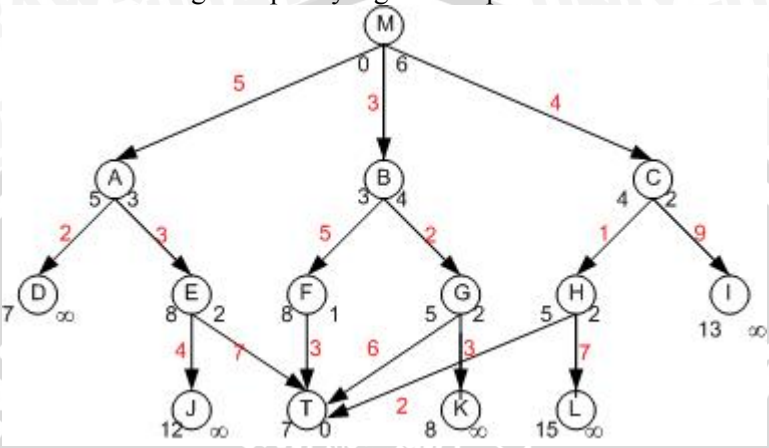
$$f'(x) = g(x) + h'(x) \tag{2.11}$$

Beberapa istilah yang digunakan dalam algoritma ini antara lain *open list, closed list, start node, goal node* dan *parent. Open list* diumpamakan seperti daftar belanjaan. Dalam algoritma ini *open list* berisi daftar titik (*node*) yang memiliki kemungkinan untuk dikunjungi berikutnya. Dengan kata lain di dalamnya terdapat *node-node* yang harus diperiksa. Jika dari *open list* sudah ditemukan titik yang dipilih sebagai jalur berikutnya maka bisa dimasukkan dalam *closed list*. Dengan *closed list* ini bisa diketahui *node-node* yang telah diperiksa. Pada kasus perjalanan mobil patroli ini *start node* adalah titik rawan awal yang digunakan sebagai titik berangkat semua mobil patroli. *Goal node* adalah *node* tujuan untuk masing-masing mobil patroli. *Parent* adalah *node* yang dikunjungi tepat sebelum sebuah *node*.

Berikut ini *pseudo-code* dari algoritma A*:

```
function A*(start,goal)
    var closed := the empty set
    var q := make_queue(path(start))
    while q is not empty
        var p := remove_first(q)
        var x := the last node of p
        if x in closed
            continue
        if x = goal
            return p
        add x to closed
        foreach y in successors(p)
            enqueue(q, y)
    return failure
```

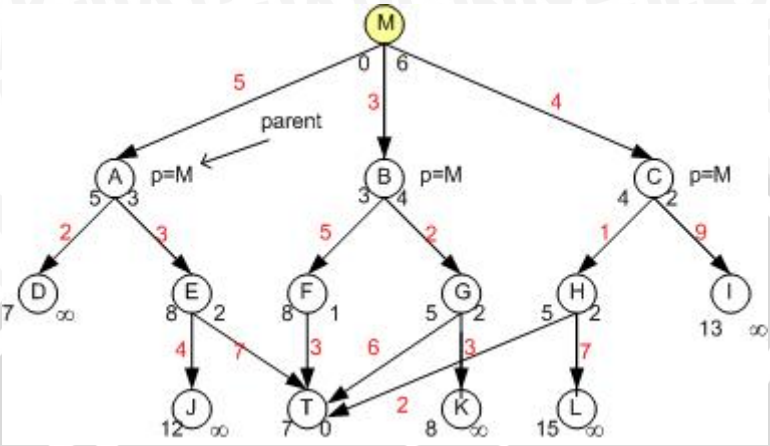
Untuk lebih jelasnya, berikut ini adalah contoh penggunaan algoritma A* untuk graf seperti yang terlihat pada Gambar 2.5.



Gambar 2.5 Graf Contoh Pencarian Rute dengan Algoritma A*

Node M merupakan *start node* sedangkan *node* T merupakan *goal node*. Nilai *edge* dari M ke A merupakan biaya yang dibutuhkan dari *node* M menuju ke *node* A. Nilai g(x) diperoleh dari biaya *edge* minimal, sedangkan nilai h'(x) diperoleh dari perkiraan biaya yang dibutuhkan dari *node* tersebut menuju *goal node* (nilai ini tidak menjamin keberadaan solusi, hanya perkiraan saja). h'(x) bernilai ∞ jika sudah jelas tidak ada hubungan antara *node* x dengan *goal node* (jalan buntu). Misalkan untuk *node* M memiliki nilai g(x) = 0 dan h'(x) = 6, *node* A memiliki nilai g(x) = 5 dan h'(x) = 3.

Mula-mula *start node* dimasukkan dalam *closed list* kemudian dicari anak-anaknya yaitu A, B dan C. Karena ketiga *node* tersebut belum pernah masuk dalam *open list* maupun *closed list* maka ketiganya langsung dimasukkan dalam *open list* dan masing-masing diinisialisasi *parent*nya yaitu M. Antrian dalam *open list* berdasarkan nilai f'(x) dari kiri ke kanan adalah dari kecil ke besar. Warna kuning melambangkan bahwa *node* tersebut telah diuji atau telah dimasukkan dalam *closed list*. Seperti terlihat dalam Gambar 2.6.

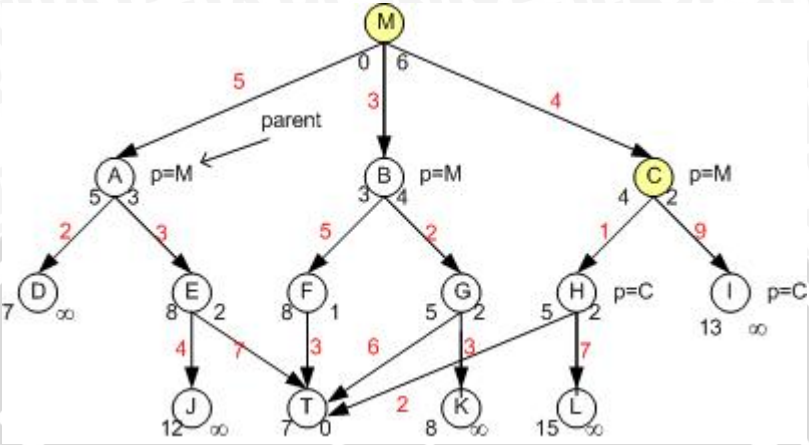


Gambar 2.6 Langkah ke-1 Pencarian Rute dengan Algoritma A*

Tabel 2.5 *Open List* dan *Closed List* pada Langkah-1

Open List	*Closed List*
{C(6), B(7), A(8)}	{M}

Selanjutnya dari antrian dalam *open list* diambil *node* dengan nilai f'(x) terkecil yaitu *node* C. *Node* tersebut kemudian dimasukkan dalam *closed list*. Lalu dilakukan pengecekan pada anak-anak C yaitu H dan I. Karena keduanya belum pernah masuk dalam *closed list* maupun *open list* maka langsung dimasukkan dalam *open list* dan *node* C menjadi *parent* H dan I.

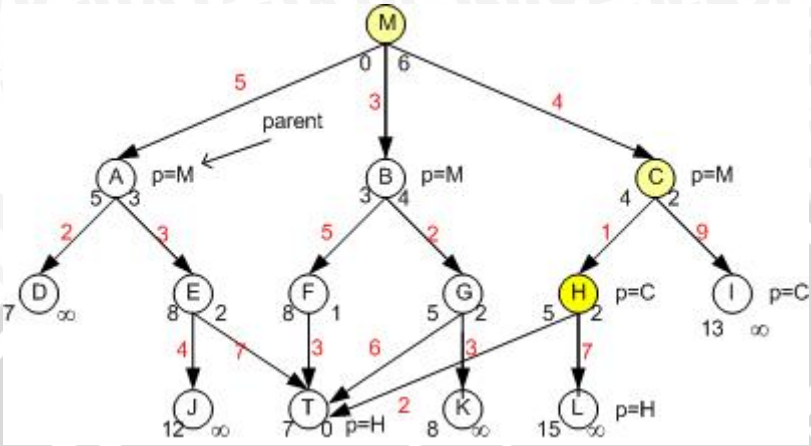


Gambar 2.7 Langkah ke-2 Pencarian Rute dengan Algoritma A*

Tabel 2.6 *Open List* dan *Closed List* pada Langkah-2

Open List	*Closed List*
{H(7), B(7), A(8), I(∞)}	{C, M}

Dari antrian yang terdapat dalam *open list* maka diambil *node* dengan nilai f'(x) paling kecil yaitu H. Selanjutnya H dimasukkan ke dalam *closed list*. Kemudian dilakukan pengecekan terhadap anak dari *node* H dan diperoleh L serta T. Karena keduanya belum pernah masuk dalam *closed list* maupun *open list* maka keduanya langsung dimasukkan dalam *open list*. Seperti yang terlihat pada Gambar 2.8 dan Tabel 2.6.



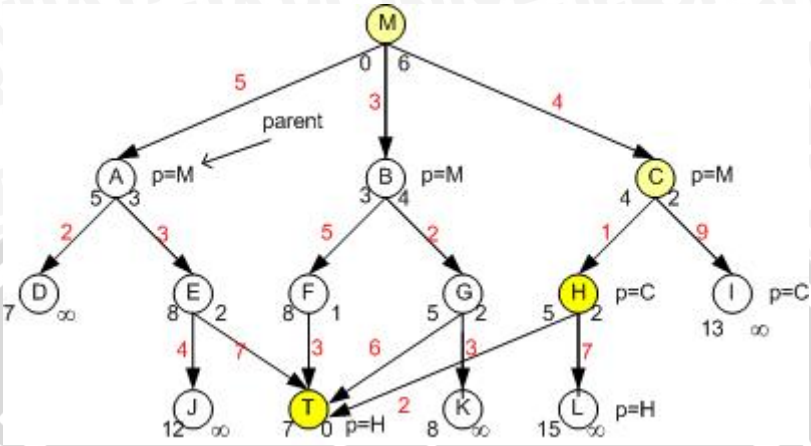
Gambar 2.8 Langkah ke-3 Pencarian Rute dengan Algoritma A*

Tabel 2.7 *Open List* dan *Closed List* pada Langkah-3

Open List	*Closed List*
{T(7), B(7), A(8), I(∞), L(∞)}	{H, C, M }

Kemudian dilakukan pengecekan lagi dari antrian yang ada dalam *open list* dan diperoleh *node* dengan nilai f'(x) paling kecil yaitu T. Kemudian T dimasukkan ke dalam *closed list* seperti diperlihatkan dalam Gambar 2.9 dan Tabel 2.8.

Karena T merupakan *goal node* maka proses berhenti. Dan untuk mendapatkan jalurnya bisa dilakukan *backward*. Yaitu proses untuk mencari jalur dari *goal node* sampai ke *start node* dengan melihat *parent*nya. Pada kasus ini proses *backward* dimulai dari *node* T. *Node* T mempunyai *parent* H, *node* H mempunyai *parent* C, *node* C mempunyai *parent* M. Karena M merupakan *start node* maka proses *backward* berhenti. Dan diperoleh jalurnya adalah M-C-H-T.



Gambar 2.9 Langkah ke-4 Pencarian Rute dengan Algoritma A*

Tabel 2.8 Antrian *Open* dan *Closed* pada Langkah-4

Antrian *Open*	Antrian *Closed*
{B(7), C(8), I(∞), L(∞)}	{T, H, C, M}

2.4 Basis Data

Masukan yang digunakan dalam sebuah aplikasi biasanya berupa data. Data sendiri adalah sekumpulan deskripsi dari benda-benda dan kejadian-kejadian. Atau sekumpulan dari fakta-fakta.

Untuk data dengan ukuran kecil, kita bisa saja dengan mudah melakukan pengolahan terhadap data tersebut. Namun jika data yang harus kita olah berukuran besar bahkan sangat besar, maka akan timbul suatu permasalahan untuk mengolah data tersebut. Untuk itu data tersebut perlu untuk disimpan dalam suatu basis data. Secara singkat basis data digunakan untuk menyimpan data agar data tersebut dapat dimanipulasi dengan mudah, terjamin keakuratannya, efisien dalam penyimpanannya dan tentu saja dapat dengan mudah diakses kembali (Bambang Robi'in, 2005:2). Untuk melakukan pengolahan data dalam basis data tersebut digunakan bahasa *query* standar, salah satunya adalah *Structure Query Language (SQL)*.

2.5 Sistematika Perjalanan Mobil Patroli Keamanan Polisi

Pelaksanaan patroli keamanan polisi sudah dilaksanakan sejak dulu. Untuk lebih mengetahui sistematika perjalanan mobil patroli keamanan polisi yang sebenarnya, maka dilakukan survey patroli keamanan di wilayah Polresta Madiun.

Pada tingkat kota atau Polresta, patroli dibagi menjadi dua bagian yaitu patroli terbuka dan patroli tertutup. Patroli terbuka adalah patroli yang dilakukan oleh anggota kepolisian bagian lalu lintas dan SABARA (Samapta Bayangkara). Di Polresta Madiun sendiri untuk patroli terbuka ini digunakan beberapa kendaraan yang memiliki fungsi yang berbeda-beda. Selain kendaraan-kendaraan tersebut, masih ada satu lagi patroli terbuka yang dilakukan oleh Polresta yaitu SABARA. Patroli ini hanya mengunjungi tempat-tempat vital di kota. Seperti misalnya perusahaan-perusahaan besar, bank, instansi pemerintahan dan lain-lain.

Sedangkan pada patroli tertutup yang dilaksanakan oleh bagian Intel dan Serse, lebih ditujukan untuk pengawasan orang asing, teroris, atau adanya kegiatan yang dicurigai mengarah pada tindak kriminal. Jadi pelaksanaan patroli tertutup ini hanya sewaktu-waktu jika ada perintah untuk melakukan pengawasan terhadap sesuatu atau seseorang yang dianggap mencurigakan.

Pada tingkat kecamatan atau Polsekta di seluruh wilayah kota Madiun juga telah dilakukan patroli dalam skala kecil yaitu hanya melewati jalan-jalan dalam satu wilayah kecamatan tersebut.

Dalam tugas akhir ini penulis mengambil kasus pelaksanaan patroli keamanan polisi terbuka di wilayah kota. Hal ini mengingat untuk patroli keamanan di wilayah Polsekta, rute yang dilalui masih terlalu kecil. Khususnya penulis mengambil kasus patroli keamanan yang dilakukan oleh SABARA. Dalam teorinya, semua patroli wajib dilakukan secara terus-menerus baik pagi, siang maupun malam. Namun mengingat adanya keterbatasan biaya operasional untuk kendaraan dan keterbatasan manusia, maka patroli hanya dilakukan sesekali. Itu pun dengan keterbatasan masih ada beberapa tempat yang tidak sempat dikunjungi. Hal ini juga terjadi pada pelaksanaan patroli yang dilakukan oleh SABARA.

SABARA biasanya melakukan patroli hanya sekali dalam sehari melalui rute yang telah ditentukan. Jadi dalam sekali patroli, tidak semua tempat-tempat vital bisa dikunjungi. Hal ini memungkinkan terjadinya tindak kriminal pada tempat-tempat tidak dikunjungi

karena tidak adanya pengawasan yang lebih intensif dari pihak kepolisian, hanya ada pengawasan dari pihak keamanan yang berjaga di tempat tersebut. Karena kelemahan-kelemahan tersebut, maka pelaksanaan patroli keamanan belum bisa mencapai hasil yang maksimal.

2.6 Pencarian Rute Minimum dengan Menghitung Semua Kemungkinan

Untuk mengetahui nilai pencarian rute yang paling minimum bisa dilakukan dengan cara menghitung semua kemungkinan yang ada. Langkah-langkah perhitungan semua kemungkinan rute ini seperti dijelaskan sebagai berikut:

1. Dilakukan pendataan rute-rute yang mungkin untuk dilalui.
2. Kemudian dilakukan perhitungan jarak yang diperoleh dari masing-masing kemungkinan rute yang ada.
3. Selanjutnya jarak dari masing-masing kemungkinan rute yang telah dihitung tersebut dibandingkan untuk diambil jarak yang paling minimum.
4. Jarak paling minimum inilah yang menjadi hasil pencarian rute minimum dengan menghitung semua kemungkinan.

2.7 Analisis Galat

Dalam sub bidang ilmu metode numerik dikenal istilah galat (*error*). Galat adalah selisih antara solusi sebenarnya (*exact solution*) dengan solusi hampiran (*approximation*). Galat berasosiasi dengan seberapa dekat solusi hampiran terhadap solusi sebenarnya. Semakin kecil galatnya, semakin teliti solusi numerik yang didapatkan (Rinaldi Munir, 2003:23).

Selisih (galat) antara nilai sebenarnya dengan nilai hampiran ditunjukkan dalam Persamaan 2.12.

$$\varepsilon = a - \hat{a} \qquad (2.12)$$

Keterangan :

ε : galat
a : nilai sebenarnya
$\hat{a}$: nilai hampiran

Karena nilai positif atau negatif tidak dipertimbangkan, maka galat didefinisikan sebagai suatu nilai mutlak seperti ditunjukkan dalam Persamaan 2.13.

$$|\varepsilon| = |a - \hat{a}| \tag{2.13}$$

Dari Persamaan 2.12 dan Persamaan 2.13 tidak terdapat informasi seberapa besar galat yang terjadi pada nilai hampiran dibandingkan dengan nilai sebenarnya. Sehingga bisa menimbulkan nilai galat yang rancu untuk kasus yang berbeda. Misalkan seorang anak mengatakan bahwa panjang suatu kawat adalah 99 cm padahal panjang sebenarnya adalah 100 cm maka galatnya adalah 1 cm. Pada kasus lain seorang anak mengatakan panjang pensilnya adalah 9 cm padahal panjang sebenarnya adalah 10 cm maka galatnya juga sama dengan kasus pertama yaitu 1 cm. Jika digunakan Persamaan 2.12 atau Persamaan 2.13, mungkin kedua galat tersebut akan dianggap sama saja. Untuk menghindari kerancuan interpretasi nilai galat, maka nilai galat tersebut harus dinormalkan terhadap nilai sebenarnya dan disebut sebagai galat relatif, seperti ditunjukkan dalam Persamaan 2.14 dan Persamaan 2.15.

$$\varepsilon_R = \frac{\varepsilon}{a} \tag{2.14}$$

ε_R : galat relatif

Atau dalam bentuk presentase

$$\varepsilon_R = \frac{\varepsilon}{a} \text{ x } 100\% \tag{2.15}$$

BAB III
METODOLOGI DAN PERANCANGAN

3.1 Analisis Sistem

3.3.1 Deskripsi Umum Sistem

Sistem pencarian rute terpendek *multi Travelling Salesman Problem (m-TSP)* untuk kasus mobil patroli keamanan polisi ini dibuat pada sebuah komputer dengan spesifikasi sebagai berikut:

1. Prossesor 1.79 Ghz
2. Memori 256 MB
3. Sistem operasi Microsoft Windows XP Professional
4. Database Microsoft Access 2003
5. Bahasa Pemrograman Delphi 6.0

Kerja sistem dibagi menjadi dua bagian besar yaitu proses penginisialisasian peta dan pencarian rute. Adapun tahap-tahap penginisialisasian peta adalah sebagai berikut:

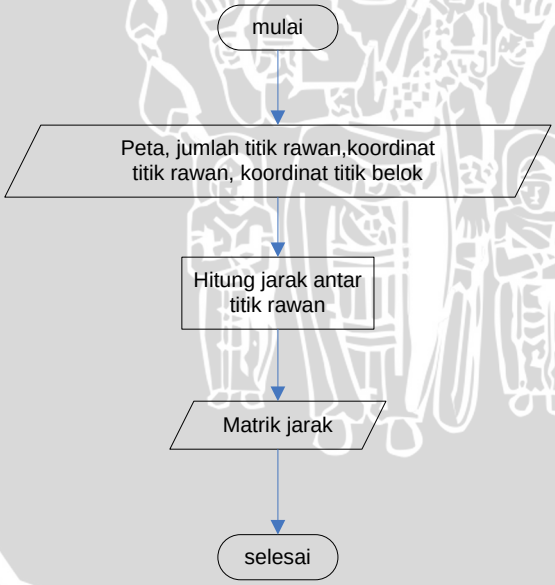
1. Peta yang akan diinisialisasi dimasukkan terlebih dahulu. Bisa peta yang belum ada dalam sistem atau peta yang sudah ada kemudian diinisialisasi ulang.
2. Kemudian titik-titik kerawanan ditentukan koordinatnya dan dicari jaraknya. Misalkan untuk menentukan jarak antara titik T1 dengan T2. Mula-mula pengguna meng-klik pada titik T1 kemudian jika bertemu dengan suatu belokan maka pengguna harus meng-klik titik belok tersebut. Sistem akan menghitung jarak T1 sampai titik belok tersebut (misalkan d1). Lalu pengguna meng-klik titik belok berikutnya dan menghitung jarak titik belok pertama sampai titik belok kedua kemudian menjumlahkannya dengan d1. Begitu seterusnya sampai pengguna meng-klik T2.
3. Hasil jarak titik kerawanan tersebut disimpan oleh sistem dalam *database* beserta nama peta dan koordinat titik-titik kerawanan.

Sedangkan berikut ini adalah tahap-tahap pencarian rute:

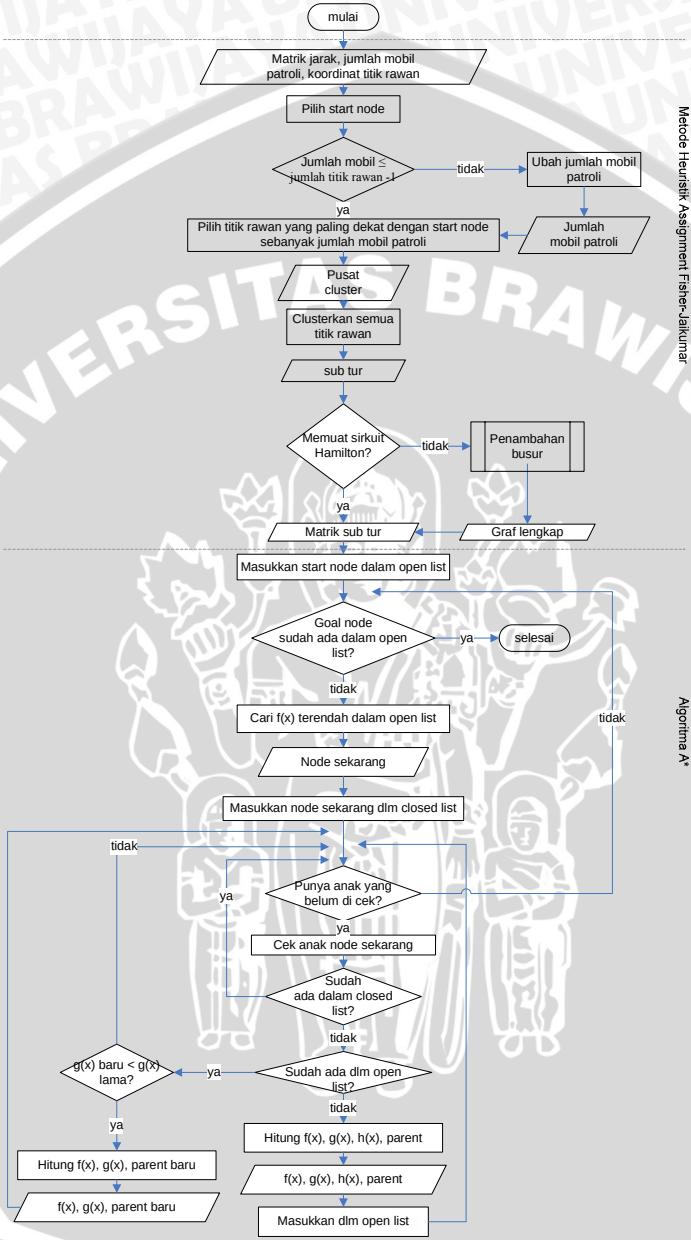
1. Masukan pada sistem diperoleh dari pengguna, yaitu dengan cara pengguna memilih peta apa yang ingin dianalisanya, jumlah daerah rawan serta jumlah mobil yang ingin dipergunakan.

2. Sistem yang menerima masukan tersebut kemudian akan mengambil *database* yang dimiliki oleh peta tersebut yaitu data matrik jarak dan koordinat titik kerawanan yang dimiliki oleh peta tersebut.
3. Dari data tersebut kemudian dilakukan pengolahan dalam sistem dengan menggunakan algoritma *heuristic* Fisher-Jaikumar sehingga terbentuklah beberapa sub tur.
4. Pada masing-masing sub tur dilakukan optimasi pencarian rute terpendek dengan algoritma A*.
5. Hasil akhir dari pengolahan tersebut berupa total jarak yang ditempuh serta rute terpendek yang dihasilkan. Hasil tersebut diolah dulu menjadi graf baru kemudian hasilnya dikembalikan pada pengguna.

Untuk lebih jelasnya mengenai proses inisialisasi peta dan pencarian rute, pada Gambar 3.1 diberikan gambar diagram alir dari proses inisialisasi peta dan Gambar 3.2 yang merupakan diagram alir proses pencarian rute.



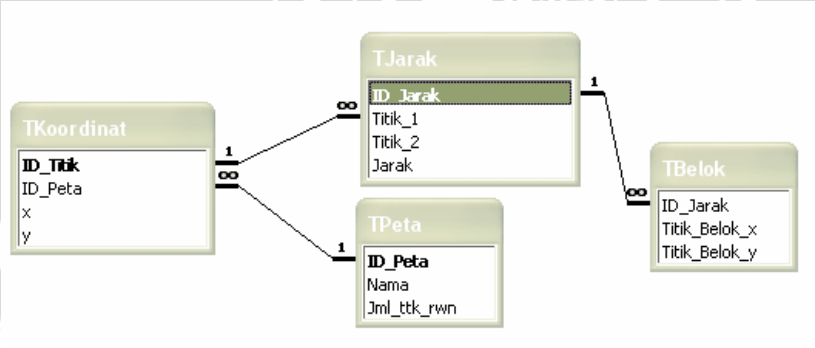
Gambar 3.1 Diagram Alir Penginisialisasian Peta



Gambar 3.2 Diagram Alir Pencarian Rute

3.2 Rancangan Basis Data

Pada sistem ini akan dibangun basis data menggunakan Microsoft Access. Basis data dalam sistem ini terdiri dari empat tabel yaitu tabel TPeta, TKoordinat, TJarak dan TBelok. Tabel TPeta digunakan untuk menyimpan ID_Peta serta nama peta apa saja yang ada dalam *database*. Tabel TKoordinat digunakan untuk menyimpan titik-titik rawan yang ada pada masing-masing peta. Tiap titik akan diberi ID yang unik sehingga menjamin tidak adanya ID yang sama antar titik rawan baik yang masih dalam satu peta maupun antar peta yang berbeda. Tabel ketiga yaitu tabel TJarak, digunakan untuk menyimpan jarak antar titik rawan dalam satu peta. Sedangkan tabel yang terakhir yaitu tabel TBelok digunakan untuk menyimpan koordinat x dan koordinat y dari titik belok pada suatu jarak. Satu ID_Jarak bisa memiliki satu atau lebih titik belok. Dengan asumsi jika tidak terdapat jalur yang menghubungkan antar dua titik rawan dalam satu peta, maka jarak antar titik tersebut adalah 0. Hubungan antar tabel dalam sistem ini seperti terlihat dalam Gambar 3.3.



Gambar 3.3 Skema Basis Data

Definisi tabel beserta atributnya ditunjukkan dalam Tabel 3.1, Tabel 3.2, Tabel 3.3 dan Tabel 3.4.

Tabel 3.1 Tabel TPeta

Field	Tipe Data	Keterangan
ID_Peta	Integer	Primary key, not null
Nama	Text (30)	

Keterangan:
ID_Peta : berisi ID peta
Nama : nama peta

Tabel 3.2 Tabel TKoordinat

Field	Tipe Data	Keterangan
ID_Titik	Text (10)	Primary key, not null
ID_Peta	Integer	
Titik	Integer	
X	Integer	
Y	Integer	

Keterangan:
Titik : berisi bilangan integer 1, 2, …, n
Di mana n adalah jumlah titik rawan dalam peta tersebut.
X : posisi koordinat x titik rawan tersebut dalam peta
Y : posisi koordinat y titik rawan tersebut dalam peta

Tabel 3.3 Tabel TJarak

Field	Tipe Data	Keterangan
ID_Jarak	Text(30)	Primary key, not null
Titik_1	Text (10)	not null
Titik_2	Text (10)	not null
Jarak	Integer	

Keterangan:
Titik_1 : berisi ID_Titik pertama yang dicari jaraknya
Titik_2 : berisi ID_Titik kedua yang dicari jaraknya
Jarak : nilai jarak dari titik pertama ke titik kedua

Tabel 3.4 Tabel TBelok

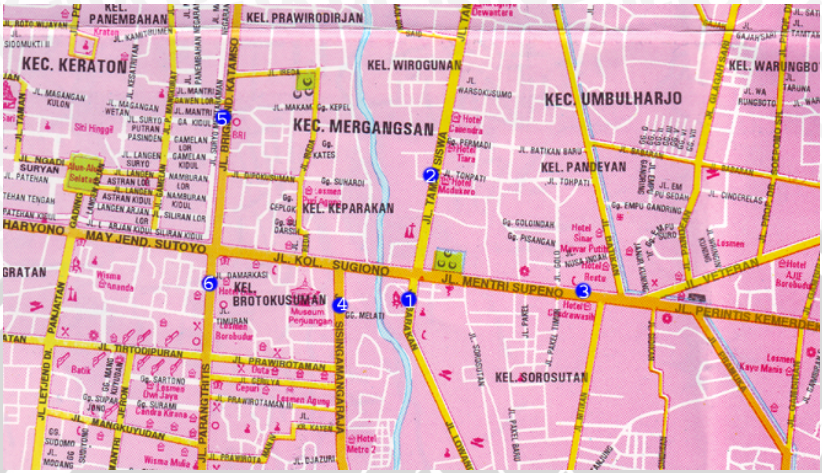
Field	Tipe Data	Keterangan
ID_Jarak	Text(30)	
Titik_Belok_x	Integer	
Titik_Belok_y	Integer	

Keterangan:
Titik_Belok_x : berisi koordinat x dari titik belok
Titik_Belok_y : berisi koordinat y dari titik belok

3.3 Inisialisasi Data

Inisialisasi data dalam sistem ini adalah suatu proses untuk menginisialisasi peta beserta titik rawan yang dimilikinya. Proses ini bisa untuk menginisialisasi peta yang sebelumnya belum ada dalam *database* maupun menginisialisasi ulang (*mengupdate*) peta yang sebelumnya sudah ada dalam *database*. Hasil dari proses ini yaitu berupa data yang disimpan dalam tabel TPeta, TKoordinat, TJarak dan TBelok. Untuk lebih jelasnya, langkah-langkah penginisialisasian peta ini akan dijelaskan beserta dengan contoh.

Berikut ini diberi masukan pada proses inisialisasi ini berupa peta kota Jogja dengan jumlah titik rawan sebanyak enam (dengan asumsi satu kantor polisi dan lima titik rawan). Pengguna kemudian memberi tanda posisi titik-titik rawan tersebut seperti terlihat pada Gambar 3.4. Titik rawan (1) diasumsikan sebagai kantor polisi. Sedangkan titik-titik rawan yang lain merupakan tempat-tempat vital yang ada dalam kota seperti misalnya perusahaan besar, bank, hotel, instansi pemerintahan dan tempat-tempat vital lainnya. Dari Gambar 3.2 bisa dilihat bahwa titik rawan (1) adalah kantor polisi sebagai tempat berangkat semua mobil patroli, titik rawan (2), (3) dan (6) adalah hotel, titik rawan (4) adalah Museum Perjuangan dan titik rawan (5) adalah bank.



Gambar 3.4 Peta Kota Jogja beserta Titik-titik Rawannya

Nama peta tersebut kemudian disimpan dalam tabel TPeta. Sedangkan posisi masing-masing titik rawan disimpan dalam tabel TKoordinat. Posisi masing-masing titik rawan ditunjukkan dalam Tabel 3.5.

Tabel 3.5 Koordinat Titik Rawan

Titik Rawan	Koordinat X	Koordinat Y
1	394	912
2	414	810
3	537	907
4	339	918
5	243	764
6	231	901

Selanjutnya dilakukan proses inisialisasi jarak antar titik rawan yaitu dengan cara pengguna menginisialisasi titik-titik belok yang ada di antara dua titik rawan yang dicari jaraknya.

Gambar 3.5 Inisialisasi Jarak Titik 1 Sampai Titik 2

Dari Gambar 3.5 kita bisa mengetahui bahwa antara titik rawan (1) dan (2) tidak terdapat titik belok sehingga sistem akan langsung menghitung jarak antara titik (1) dan (2) menggunakan koordinat masing-masing titik yaitu berdasarkan Persamaan 3.1.

$$\text{Jarak} = \sum_{i=1}^{n} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{3.1}$$

Keterangan :

n : jumlah titik belok antar dua titik rawan tambah satu.

i : titik rawan dan titik belok, dimulai dari titik dimulainya pencarian jarak, *i* bertambah tiap kali bertemu dengan titik belok.

Jadi bisa dihitung

$$\text{Jarak titik (1) dan (2)} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
$$= \sqrt{(414 - 394)^2 + (810 - 912)^2}$$
$$= \sqrt{400 + 10404}$$
$$= 103{,}94$$
$$\approx 104$$

Keterangan :
x_1, y_1 : koordinat titik rawan (1)
x_2, y_2 : koordinat titik rawan (2)

Karena untuk jarak dari titik (2) ke titik (1) melalui jalur yang sama dalam peta maka jarak dari titik (2) ke titik (1) sama dengan jarak dari titik (1) ke titik (2).

Selanjutnya dilakukan penghitungan jarak pada peta dari titik (1) ke titik (3). Seperti pada langkah di atas, terlebih dahulu dilakukan inisialisasi titik belok antar dua titik tersebut seperti dalam Gambar 3.6.

Gambar 3.6 Inisialisasi Jarak Titik 1 Sampai Titik 3

Jarak antara titik (1) dan titik (3) bisa diperoleh dengan cara yang sama seperti mencari jarak antara titik (1) dan titik (2).

Jarak titik 1 dan 3 = $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2}$
= 162

Keterangan :
x_1, y_1 : koordinat titik rawan 1
x_2, y_2 : koordinat titik belok pertama
x_3, y_3 : koordinat titik rawan 3.

Untuk jarak antar titik-titik rawan lain bisa dihitung seperti cara di atas sehingga diperoleh hasil seperti pada Tabel 3.6.

Tabel 3.6 Jarak Antar Titik rawan

Titik Rawan	1	2	3	4	5	6
1	0	104	162	122	295	220
2	104	0	220	179	351	281
3	162	220	0	239	409	341
4	122	179	239	0	240	167
5	295	351	409	240	0	138
6	220	281	341	167	138	0

Kolom 1 pada Tabel 3.6 menunjukkan jarak antara titik rawan 1 dengan titik-titik rawan lain, kolom 2 menunjukkan jarak antara titik rawan 2 dengan titik rawan lainnya, begitu seterusnya. Misalkan untuk kolom 1 baris 2 terdapat angka 104 yang merupakan jarak dari titik rawan 1 menuju ke titik rawan 2.

3.4 Pencarian Rute

Setelah semua jarak antar titik rawan diketahui maka akan terbentuk matrik jarak seperti dalam Tabel 3.6. Pada kasus ini diasumsikan jalur antar *node* merupakan jalan dengan dua arah. Berarti jika diberikan *node* 1 dan *node* 2, maka jarak *node* 1 ke *node* 2 sama dengan jarak *node* 2 ke *node* 1. Pada kenyataannya nanti jarak *node* 1 ke *node* 2 belum tentu sama dengan jarak *node* 2 ke *node* 1. Kemudian diberi masukan pada sistem berupa jumlah mobil patroli yang digunakan dalam patroli keamanan tersebut.

```
Jumlah mobil patroli = 2
```

Berdasarkan masukan dapat diketahui bahwa

```
jumlah mobil < jumlah titik rawan, sehingga
jumlah sub tur = jumlah mobil = 2
```

Selanjutnya dibuat sub tur menggunakan metode *heuristic assignment* dengan algoritma Fisher-Jaikumar. Mula-mula dilakukan pencarian tiga titik rawan yang mempunyai jarak paling dekat dengan kantor polisi (*node* pusat) yang kemudian dijadikan sebagai titik pusat sub tur.

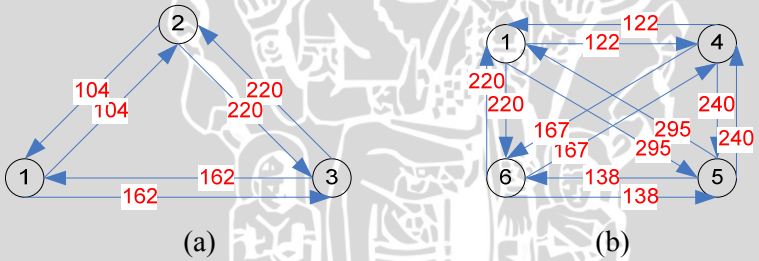
```
Node pusat =  node 1
Sub tur 1
pusat sub tur      =  node 2
Sub tur 2
pusat sub tur      =  node 4
```

Kemudian untuk *node-node* yang belum terpilih yaitu *node* 3, 5 dan 6, dicari jarak paling dekat dengan sub tur yang mana. Lalu masukkan dalam anggota sub tur tersebut.

```
Sub tur 1   =  node 3
Sub tur 2   =  node 5, 6
```

Dari langkah di atas dihasilkan dua sub tur yang bisa direpresentasikan dalam graf berarah seperti dalam Gambar 3.7.

Dari Gambar 3.7 kita bisa mengetahui bahwa graf pertama dan graf kedua sudah merupakan graf Hamilton maka baik graf pertama maupun graf kedua tidak perlu dilakukan penambahan busur.



(a) (b)

Gambar 3.7 (a) Representasi Sub Tur 1; (b) Representasi Sub Tur 2

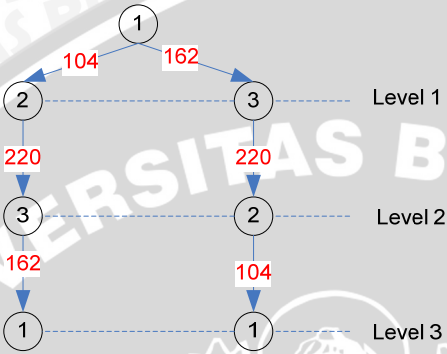
Langkah berikutnya pada masing-masing graf (sub tur) dilakukan optimasi menggunakan algoritma A*. Terdapat beberapa fungsi yang digunakan dalam perhitungan yaitu

f'(x) = fungsi eveluasi yang diperoleh dari g(x) + h'(x)

g(x) = biaya yang sudah dikeluarkan dari *node* awal sampai *node* x

h'(x) = perkiraan biaya untuk sampai *goal node* dari *node* x

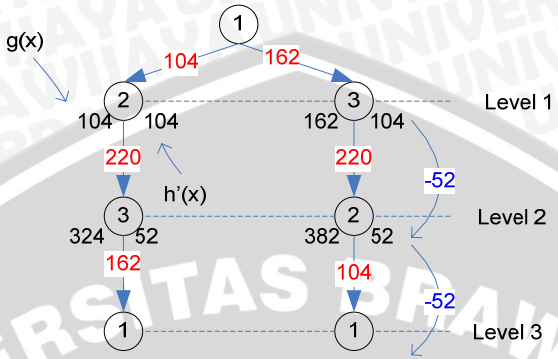
Untuk mempermudah optimasi, maka graf pada masing-masing sub tur diubah terlebih dahulu menjadi bentuk *tree*. Bentuk *tree* dari sub tur 1 seperti dalam Gambar 3.8.



Gambar 3.8 Bentuk *Tree* Sub Tur 1

Dengan dimodifikasi menjadi *tree,* menjamin tidak adanya titik rawan yang tidak dikunjungi oleh mobil patroli. Di mana tiap *node* pada tiap level dianggap sebagai *node* yang berbeda. Misalkan *node* 2 pada level pertama merupakan *node* yang berbeda dengan *node* 2 yang ada pada level kedua. Untuk menentukan nilai h'(x) masing-masing *node* berdasarkan nilai jarak terendah dalam matrik jarak sub tur tersebut kemudian dibagi dengan jumlah level kurang satu. Jumlah level sendiri sama dengan jumlah *node* yang ada dalam sub tur tersebut. Khusus untuk level paling bawah nilai h'(x) selalu nol karena diasumsikan jika suatu jalur bisa mencapai level paling bawah maka jalur tersebut bisa mencapai *node* tujuan dan telah mengunjungi semua *node*. Sedangkan node yang menjadi akhir suatu jalur namun bukan *goal node* (jalan buntu) memiliki nilai h'(x) = ∞. Untuk lebih jelasnya perhitungan h'(x) pada sub tur 1 akan dijelaskan dalam Gambar 3.9.

```
Nilai jarak terendah dari sub tur 1 = 104
Jumlah level = jumlah node = 3
Pengurangan h'(x)
= Nilai jarak terendah : (jumlah level-1)
= 104 : (3-1)
= 52
```



Gambar 3.9 Inisialisasi g(x) dan h'(x) pada Sub Tur 1

Selanjutnya *closed list* diisi dengan *start node*, karena *start node* sudah diketahui yaitu node (1).

```
Closed list = {1}
Open list = {}
```

Node (1) kemudian dicari anak-anaknya, masing-masing anak dilakukan pengecekan kemudian dihitung nilai f'(x) seperti ditunjukkan pada Gambar 3.10.

```
Anak pertama :
f'(2¹)    = g(2¹) + h'(2¹)
          = 104 + 104
          = 208
```

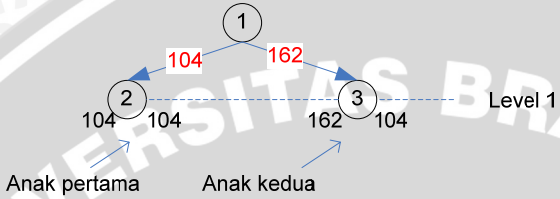
Karena *node* (2^1) (*node* 2 level 1) belum ada dalam *closed list*, maka *node* tersebut langsung dimasukkan dalam *open list* dan *node* (1) menjadi *parent*nya.

```
Anak kedua :
f(3¹)     = g(3¹) + h'(3¹)
          = 162 + 104
          = 366
```

Node (3^1) juga belum ada dalam *open list* sehingga *node* ini langsung dimasukkan ke dalam *open list* dan *node* (1) menjadi *parent node* tersebut.

Setelah semua anak selesai diperiksa dan dihitung nilai f'(x)-nya, maka isi *open list* berubah sesuai dengan urutan nilai f'(x) dari kecil ke besar.

Open list = {2^1, 3^1}



Gambar 3.10 *Node* yang Sudah Diperiksa

Selanjutnya *node* dengan nilai f'(x) terkecil pada *open list* diambil dan di masukkan ke dalam *closed list*. Sehingga *closed list* menjadi

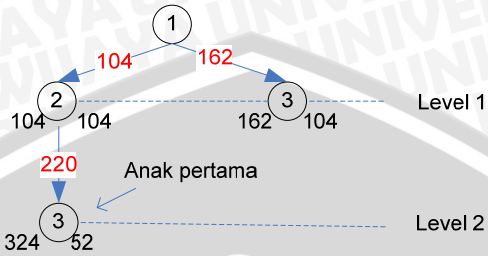
Closed list = {1, 2^1}

Node (2^1) kemudian dicari anak-anaknya dan diperiksa satu-persatu seperti dalam Gambar 3.11.

Anak pertama :
$f(3^2)$ = $g(3^2)$ + $h'(3^2)$
= 324 + 52
= 376

Karena *node* (3^2) belum ada dalam *open list*, *node* (3^2) dimasukkan dalam *open list* dan *node* (2^1) menjadi *parent node* (3^2) sehingga *open list* sekarang berubah menjadi

Open list = {3^1, 3^2}



Gambar 3.11 *Node* yang Sudah Diperiksa

Kemudian dari *open list* tersebut, *node* dengan nilai f'(x) terkecil dimasukkan ke dalam *closed list*.

$Closed\ list = \{1, 2^1, 3^1\}$

Sama seperti cara di atas, pada *node* (3^1) dilakukan pengecekan terhadap anak-anaknya dan dihitung nilai f'(x) masing-masing anak seperti terlihat pada Gambar 3.12.

Anak pertama :

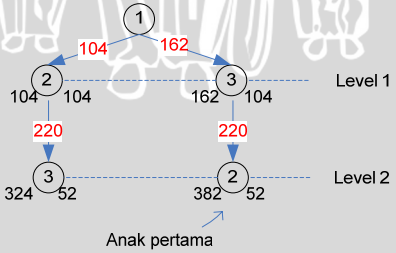
$f(2^2) = g(2^2) + h'(2^2)$

$= 382 + 52$

$= 434$

Karena *node* (2^2) belum ada dalam *open list* maka *node* tersebut langsung dimasukkan dalam *open list* dan *node* (3^1) menjadi *parent*nya.

$Open\ list = \{3^2, 2^2\}$



Gambar 3.12 *Node* yang Sudah Diperiksa

Sama seperti sebelumnya, *node* dalam *open list* dengan nilai f'(x) terkecil diambil dan dimasukkan ke dalam *closed list*.

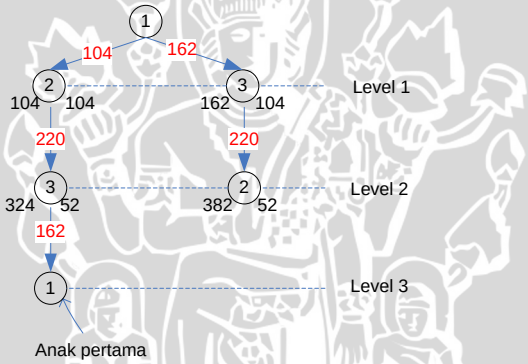
Closed list = {1, 2^1, 3^1, 3^2}

Lalu dilakukan pengecekan anak dari *node* (3^2) seperti pada Gambar 3.13.

Anak pertama :
f(1^3) = g(1^3) + h'(1^3)
= 486 + 0
= 486

Karena *node* (1^3) belum ada dalam *open list*, *node* tersebut langsung dimasukkan dalam *open list* dan *node* (3^2) menjadi *parent*nya.

Open list = {2^2, 1^3}



Gambar 3.13 *Node* yang Sudah Diperiksa

Node (1^3) yang merupakan *goal node* sudah masuk dalam *open list* sehingga proses berhenti. Untuk memperoleh jalur pada sub tur 1 dilakukan dengan melakukan backward berdasarkan *parent* masing-masing *node*, dimulai dari *goal node*.

Node 1^3 → *parent* = 3^2
Node 3^2 → *parent* = 2^1
Node 2^1 → *parent* = 1

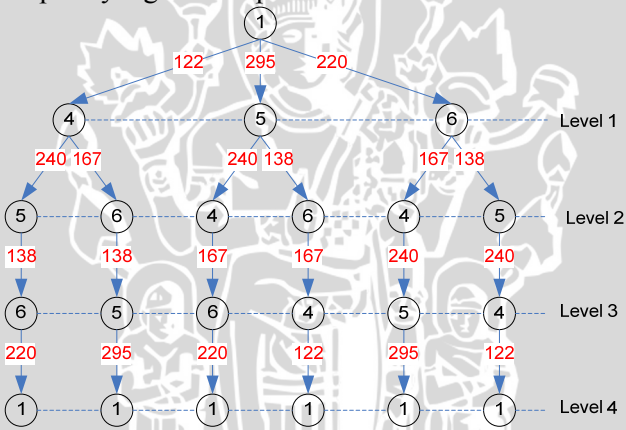
Jalurnya adalah 1 – 2^1 – 3^2 – 1^3 = 486

Untuk lebih jelasnya, alur pencarian jalur pada sub tur 1 menggunakan algoritma A* bisa dilihat pada Tabel 3.7.

Tabel 3.7 Pencarian Jalur Sub Tur 1

Node yang diambil	*Open List*	*Closed List*
		$\{1\}$
1	$\{2^1, 3^1\}$	$\{1, 2^1\}$
2^1	$\{3^1, 3^2\}$	$\{1, 2^1, 3^1\}$
3^1	$\{3^2, 2^2\}$	$\{1, 2^1, 3^1, 3^2\}$
3^2	$\{2^2, 1^3\}$	

Berikutnya dilakukan pencarian jalur pada sub tur 2. Sama seperti pada sub tur 1, sub tur 2 diubah telebih dahulu menjadi bentuk *tree* seperti yang terlihat pada Gambar 3.14.



Gambar 3.14 Bentuk *Tree* Sub Tur 2

Selanjutnya dilakukan perhitungan untuk mentukan nilai h'(x) pada masing-masing *node* dengan menggunakan cara yang sama seperti pada sub tur 1.

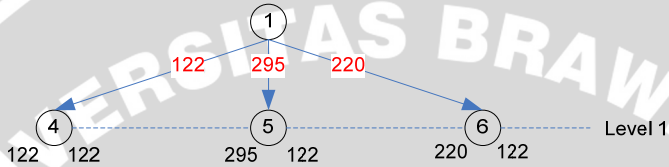
```
Nilai jarak terendah dari sub tur 2 = 122
Jumlah level = jumlah node = 4

Pengurangan h'(x)
= Nilai jarak terendah : (jumlah level-1)
```

= 122 : (4-1)
= 40,67
≈ 40

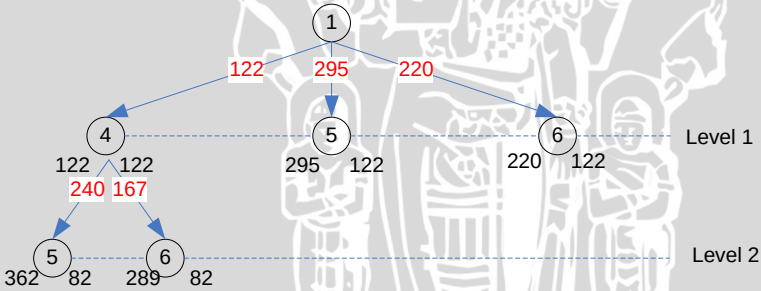
Dengan cara yang sama seperti pada sub tur 1, pada sub tur 2 ini bisa dilakukan perhitungan sebagai berikut:

Closed List = {1}



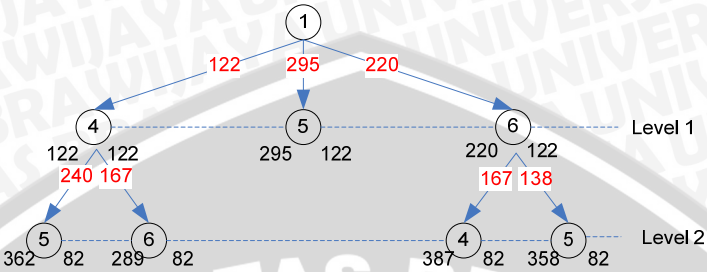
Gambar 3.15 *Node* yang Sudah Diperiksa

Open List = {4^1, 6^1, 5^1}
Parent 4^1 = 1
Parent 6^1 = 1
Parent 5^1= 1
Closed list = {1, 4^1}



Gambar 3.16 *Node* yang Sudah Diperiksa

Open List = {6^1, 6^2, 5^2, 5^1}
Parent 5^2 = 4^1
Parent 6^2 = 4^1
Closed list = {1, 4^1, 6^1}



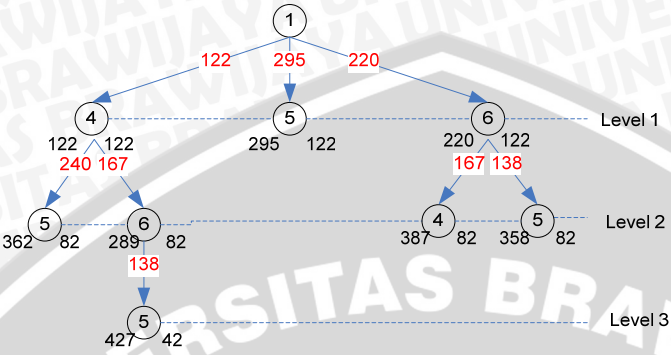
Gambar 3.17 *Node* yang Sudah Diperiksa

Seperti kita lihat pada Gambar 3.17, *node* (6^1) mempunyai anak *node* (4^2) dan (5^2). Setelah dilakukan pengecekan pada masing-masing anak, *node* (5^2) sudah ada dalam *open list*. Oleh karena itu harus dilakukan perbandingan nilai g(x) yang lama dengan yang baru. Jika g(x) baru kurang dari g(x) lama maka dilakukan perhitungan ulang nilai f(x) dengan menggunakan g(x) baru serta dilakukan perubahan *parent*. Namun jika tidak, maka nilai f(x) dan *parent* yang lama tidak perlu diubah.

$g(5^2)$ lama = 444
$g(5^2)$ baru = 440, karena
$g(5^2)$ baru < $g(5^2)$ lama

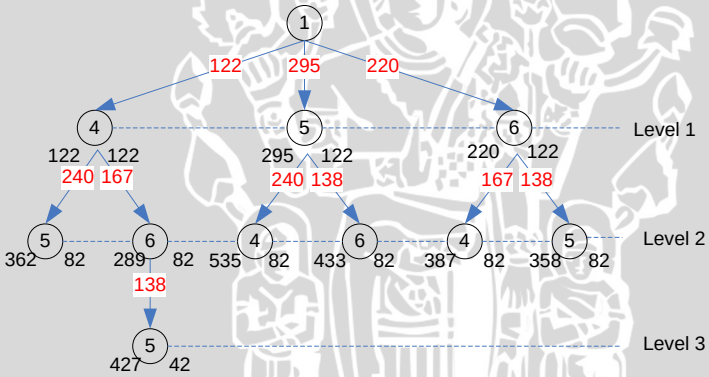
maka yang digunakan adalah $g(5^2)$ baru sehingga *open list* berubah menjadi:

Open list = {6^2, 5^1, 5^2, 4^2}
Parent 5^2 = 6^1
Closed list = {1, 4^1, 6^1, 6^2}



Gambar 3.18 *Node* yang Sudah Diperiksa

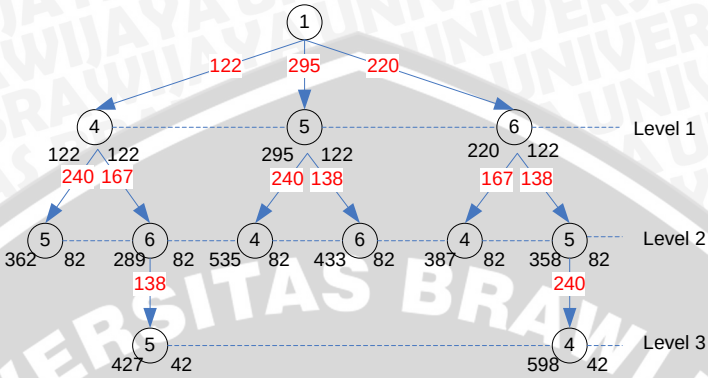
Open List = {5^1, 5^2, 5^3, 4^2}
Parent 5^3 = 6^2
Closed list = {1, 4^1, 6^1, 6^2, 5^1}



Gambar 3.19 *Node* yang Sudah Diperiksa

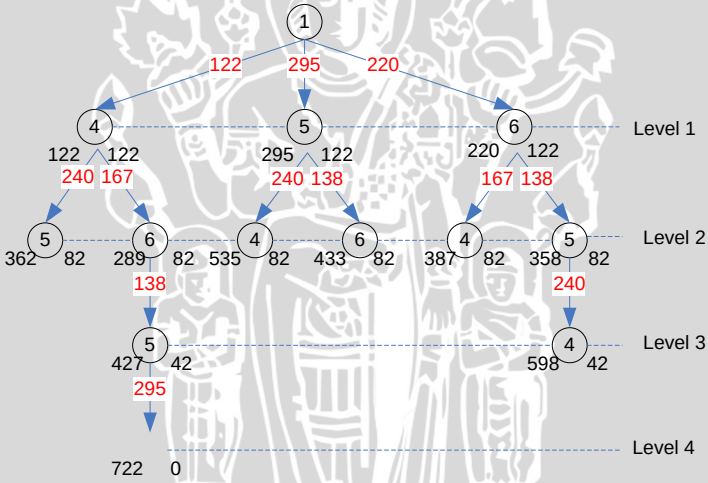
$g(4^2)$ lama = 469
$g(4^2)$ baru = 617, karena
$g(4^2)$ baru ≥ $g(4^2)$ lama
maka yang digunakan adalah $g(4^2)$ lama sehingga *open list* tetap.

Open list = {5^2, 5^3, 4^2, 6^2}
Parent 6^2 = 5^1
Closed list = {1, 4^1, 6^1, 6^2, 5^1, 5^2}



Gambar 3.20 *Node* yang Sudah Diperiksa

Open list = {5^3, 4^2, 6^2, 4^3}
Parent $4^3 = 5^2$
Closed list = {1, 4^1, 6^1, 6^2, 5^1, 5^2, 5^3}



Gambar 3.21 *Node* yang Sudah Diperiksa

Open list = {4^2, 6^2, 4^3, 1^4}
Parent $1^4 = 5^3$

Karena *goal node* sudah masuk dalam *open list*, maka proses berhenti dan selanjutnya dilakukan backward untuk menemukan jalur pada sub tur 2.

Node 1^4 → *parent* $= 5^3$

Node 5^3 → *parent* = 6^2
Node 6^2 → *parent* = 4^1
Node 4^1 → *parent* = 1

Jalurnya adalah 1 - 4^1 - 6^2 - 5^3 - 1^4 = 722

Tabel 3.8 Pencarian Jalur Sub Tur 2

Node yang diambil	*Open List*	*Closed List*
		{1}
1	{4^1, 6^1, 5^1}	{1, 4^1}
4^1	{6^1, 6^2, 5^2, 5^1}	{1, 4^1, 6^1}
6^1	{6^2, 5^1, 5^2, 4^2}	{1, 4^1, 6^1, 6^2}
6^2	{5^1, 5^2, 5^3, 4^2}	{1, 4^1, 6^1, 6^2, 5^1}
5^1	{5^2, 5^3, 4^2, 6^2}	{1, 4^1, 6^1, 6^2, 5^1, 5^2}
5^2	{5^3, 4^2, 6^2, 4^3}	{1, 4^1, 6^1, 6^2, 5^1, 5^2, 5^3}
5^3	{4^2, 6^2, 4^3, 1^4}	

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

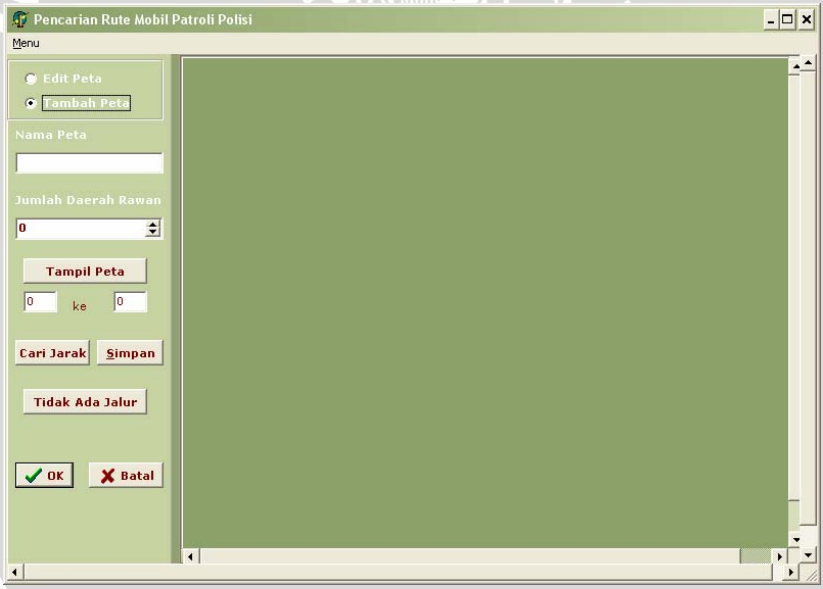
4.1 Implementasi

Sistem ini dalam penggunaannya dibagi menjadi dua bagian besar yaitu bagian untuk proses inisialisasi peta dan proses pencarian rute. Masing-masing proses memiliki fungsi serta masukan yang berbeda-beda.

4.1.1 Inisialisasi Peta

4.1.1.1 *Interface*

Interface untuk proses inisialisasi peta ditunjukkan pada Gambar 4.1.



Gambar 4.1 Tampilan Proses Inisialisasi Peta

4.1.1.2 Struktur Data

Struktur data untuk proses inisialisasi peta seperti ditunjukkan dalam Gambar 4.2.

```
type
 PBl = ^RBelok;
 RBelok = record
  x,y:integer;
 end;

 RNode = record
  x,y:integer;
  cent,invalid:boolean;
 end;
```

Gambar 4.2 Struktur Data Proses Inisialisasi Peta

Keterangan :

PBl : pointer yang menunjuk pada RBelok

RBelok : digunakan untuk menyimpan koordinat titik belok

RNode : digunakan untuk menyimpan titik-titik rawan dalam peta

Pada Proses Inisialisasi peta digunakan tipe data Rbelok dan RNode yang berupa *record* dan pointer PBl yang menunjuk pada RBelok.

4.1.1.3 Implementasi Program

Proses inisialisasi peta dimulai dengan diberikannya masukan pada sistem berupa pilihan untuk melakukan *edit* pada peta yang sudah ada atau menambah peta baru dalam *database*. Masukan berikutnya yaitu berupa nama peta, jumlah daerah rawan serta posisi dari titik rawan dan titik belok antar titik rawan.

Pada proses ini dilakukan perhitungan jarak antar titik rawan menggunakan koordinat titik rawan dan koordinat titik belok. Perhitungan dilakukan setiap kali titik belok diinisialisasi. Sedangkan untuk jarak antar titik rawan yang tidak memiliki titik belok perhitungan langsung dilakukan pada saat penyimpanan jarak.

Prosedur yang digunakan untuk melakukan proses ini ditunjukkan dalam Gambar 4.3.

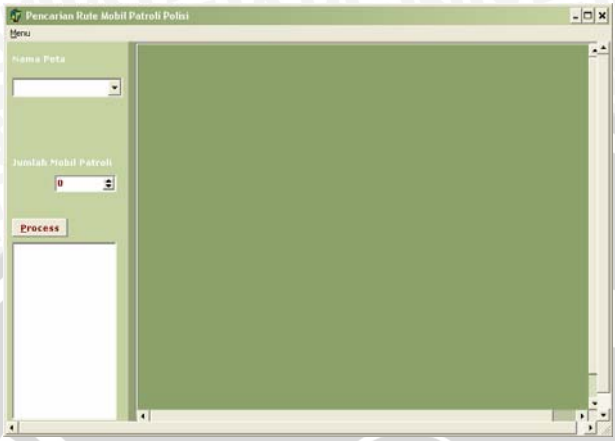
```
procedure TForm1.Image1MouseDown(Sender:
TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
Begin
    ......
    ......
 if (crJarak=1)and(d[po,pi]=0) then
 begin
  dx2:=x;
  dy2:=y;
  d1:=round(sqrt((sqr(dx2-dx1))+sqr(dy2-
     dy1)));
  dx1:=dx2;
  dy1:=dy2;
  result:=result+d1;
  //masukkan titik belok dalam List
  new(bl);
  bl^.x:=x;
  bl^.y:=y;
  blList[po,pi].Add(bl);
 end;
```

Gambar 4.3 Prosedur Perhitungan Jarak Antar Titik Rawan

4.1.2 Pencarian Rute

4.1.2.1 *Interface*

Interface dari proses pencarian rute seperti tampak dalam Gambar 4.4.



Gambar 4.4 Tampilan Proses Pencarian Rute

4.1.2.2 Struktur Data

Struktur data yang digunakan pada proses pencarian rute seperti ditunjukkan dalam Gambar 4.5.

Selain tipe data seperti yang terlihat dalam Gambar 4.5, proses pencarian rute ini juga menggunakan RBelok dan RNode yang sebelumnya telah dijelaskan dalam sub bab 4.1.1.2.

```
type
  RClust =  record
    success,valid:boolean;
    sum:integer;
  end;

  Pchild =  ^RItem;
  RItem = record
    nod:integer;
    o,c,cek:boolean;
    g,h:real;
    level:integer;
    parent:Pchild;
    child:array of Pchild;
    back:Pchild;
  end;
```

Gambar 4.5 Struktur Data Proses Pencarian Rute

Keterangan	:	
RClust	:	digunakan untuk menyimpan status serta jumlah titik rawan yang dimiliki dari masing-masing *cluster*
Pchild	:	merupakan pointer yang menunjuk pada RItem
RItem	:	digunakan untuk menyimpan titik-titik rawan setelah dimasukkan dalam *cluster*

4.1.2.3 Implementasi Program

Proses pencarian rute ini diawali dengan pengambilan data yang ada dalam *database* sesuai dengan nama peta yang telah dimasukkan oleh *user*. Selanjutnya dilakukan pemilihan pusat *cluster* untuk masing-masing sub tur serta pengklusteran titik-titik rawan ke dalam sub tur tersebut dengan asumsi titik (1) merupakan kantor polisi dan digunakan sebagai *start node* serta *goal node* untuk semua sub tur. Jumlah sub tur yang dibentuk adalah sejumlah mobil patroli yang telah dimasukkan oleh *user* dengan batasan jumlah titik rawan kurang satu. Prosedur yang digunakan untuk pemilihan pusat *cluster* seperti ditunjukkan dalam Gambar 4.6.

```
.........

for l:=1 to car do
begin
 for j:=1 to rwn do
 begin
   if
(d[1,j]<min[l])and(d[1,j]>0)and(node[j].cen
t=false)and(node[j].invalid=false) then
   begin
     min[l]:=d[1,j];
     jmin[l]:=j;
     item[l,2].nod:=j;
   end;
 end;
 node[jmin[l]].cent:=true;
end;
..........
```

Gambar 4.6 Potongan Prosedur Pencarian Pusat *Cluster*

Dari potongan prosedur dalam Gambar 4.6, dihasilkan titik-titik pusat *cluster* sebanyak jumlah mobil patroli. Masing-masing pusat *cluster* akan dijadikan sebagai acuan untuk menentukan titik-titik rawan yang lain untuk masuk dalam sub tur yang mana. Titik rawan yang akan diperiksa berikutnya adalah titik rawan yang bukan merupakan *start node* maupun pusat *cluster*. Potongan prosedur untuk menentukan anggota *cluster* ditunjukkan dalam Gambar 4.7.

```
.......

for i:=2 to rwn do
begin
 if node[i].cent=false then
 begin
  dtemp:=1000000000000;
  for k:=1 to car do
  begin
    if (d[item[k,2].nod,i]<dtemp) and
       (d[item[k,2].nod,i]>0)then
    begin
      dtemp:=d[item[k,2].nod,i];
      kmin:=k;
    end;
  end;
  inc(c[kmin].sum);
  item[kmin,c[kmin].sum].nod:=i;
end;
........
```

Gambar 4.7 Potongan Prosedur Penentuan Anggota *Cluster*

Dari potongan prosedur pada Gambar 4.7 dihasilkan sub tur-sub tur beserta masing-masing anggotanya dan jumlah anggota masing-masing sub tur dari proses pengklusteran. Selanjutnya pada masing-masing sub tur dideteksi ada tidaknya sirkuit Hamilton menggunakan prosedur seperti dalam Gambar 4.8.

```
procedure
TForm1.find_cycle(j,l,move:integer);
var k:integer;
begin
  if
    (move=c[j].sum)and(d[item[j,l].nod,1]>0)
    then c[j].success:=true
  else
  begin
    k:=1;
    while (k<=c[j].sum)  do
    begin
      if (d[item[j,l].nod,item[j,k].nod]>0)
         and
         (visited[item[j,k].nod]=false)then
      begin
       visited[item[j,k].nod]:=true;
       ham_cycle[j,k]:=item[j,k].nod;
       find_cycle(j,k,move+1);
      end;
      inc(k);
    end;
 end;
end;
```

Gambar 4.8 Prosedur Deteksi Sirkuit Hamilton

Jika hasilnya sudah ada sirkuit Hamilton maka langkah selanjutnya adalah optimasi pada sub tur tersebut, namun jika ternyata tidak ada sirkuit Hamilton, maka harus dilakukan penambahan busur terlebih dahulu pada sub tur tersebut menggunakan prosedur seperti yang ada dalam Gambar 4.9 baru kemudian dilakukan optimasi menggunakan algoritma A*.

```
procedure TForm1.addDeg(i:integer);
var j,k:integer;
begin
 for j:=1 to c[i].sum do
 begin
  for k:=1 to c[i].sum do
  begin
          if path[j,k]=false then
    begin
      aStar(i,j,k);
      d[item[i,j].nod,item[i,k].nod]:=item[
      i,k].g;
    end;
  end;
 end;
end;
```

Gambar 4.9 Prosedur Penambahan Busur

Dari prosedur penambahan busur maka dihasilkan jarak antar titik rawan yang sebenarnya tidak memiliki jalur yang menghubungkan. Berikutnya dilakukan optimasi pada masing-masing sub tur menggunakan algoritma A* seperti ditunjukkan pada Gambar 4.10, Gambar 4.11 dan Gambar 4.12.

```
procedure TForm1.aStar1(i,j,k:integer);
var m,fmin,lmin:integer;
    temp:real;
begin
 lmin:=0;
 fmin:=j;
 initTree(i);

 tree[i,fmin,lmin].c:=true;
 while (tree[i,k,c[i].sum].o=false) do
 begin
  tree[i,fmin,lmin].o:=false;
  if lmin=c[i].sum-1  then
  begin
   initChildTree(i,fmin,lmin);
   m:=1;
   while (tree[i,fmin,lmin].child[m]<>nil)
    and (m<c[i].sum) and
    (tree[i,fmin,lmin].child[m]^.cek=false)
    do
    begin
     notChild:=false;
     temp3:=tree[i,fmin,lmin].child[m];
     temp4:=@tree[i,fmin,lmin];
     cekChild;
     if
      (tree[i,fmin,lmin].child[m]^.c=false)
      and(notChild=false) then
     begin
       if
      (tree[i,fmin,lmin].child[m]^.o=false)
      then
      begin
        heuristic(i);
        tree[i,fmin,lmin].child[m]^.parent
          := @tree[i,fmin,lmin];
        tree[i,fmin,lmin].child[m]^.g
          := tree[i,fmin,lmin].g +
             d[tree[i,fmin,lmin].nod ,
          tree[i,fmin,lmin].child[m]^.nod];
        tree[i,fmin,lmin].child[m]^.h:=0;
```

Gambar 4.10 Prosedur Pencarian Rute

```
        tree[i,fmin,lmin].child[m]^.o:=true;
        tree[i,fmin,lmin].child[m]^.cek :=
           true;
      end
      else
      begin
        temp:=tree[i,fmin,lmin].child[m]^.g;
        tree[i,fmin,lmin].child[m]^.g :=
             item[i,fmin].g +
        d[tree[i,fmin,lmin].nod,
          tree[i,fmin,lmin].child[m]^.nod];
        if tree[i,fmin,lmin].child[m]^.g <
           temp then
        begin
         tree[i,fmin,lmin].child[m]^.parent^
             := tree[i,fmin,lmin];
         tree[i,fmin,lmin].child[m]^.h:=0;
         tree[i,fmin,lmin].child[m]^.cek :=
             true;
        end
        else
          tree[i,fmin,lmin].child[m]^.g :=
             temp;
      end;
    end;
    Inc(m);
   end;
 end
 else
 begin
    .......
    .......
 end;
if m=1 then
begin
 tree[i,fmin,lmin].g:=1000000000;
 tree[i,fmin,lmin].h:=100000000;
end;
```

Gambar 4.11 Lanjutan Prosedur Pencarian Rute

```
   findmin(i);
   fmin:=fx_min;
   lmin:=lev_min;
   tree[i,fmin,lmin].c:=true;
end;
```

Gambar 4.12 Lanjutan Prosedur Pencarian Rute

4.2 Penerapan Aplikasi

Untuk mengetahui optimal atau tidaknya program aplikasi ini maka dilakukan lima kali uji coba pada sistem menggunakan data sampel masing-masing sebanyak tujuh, delapan, sembilan, sepuluh dan sebelas titik rawan yang diambil dari peta kota Bandung dan Jogja, dengan asumsi titik (1) adalah kantor polisi dan titik-titik rawan yang lainnya berupa pusat-pusat keramaian yang menjadi prioritas untuk dilakukan patroli, seperti hotel, tempat industri, pusat perbelanjaan dan sebagainya. Sedangkan jumlah mobil yang digunakan sebanyak dua buah mobil patroli. Pada masing-masing uji coba diberikan koordinat titik rawan yang berbeda-beda.

Uji coba dilakukan dalam dua tahap, yang pertama yaitu dilakukan pencarian rute minimum pada data sampel menggunakan program aplikasi dengan metode *heuristic assignment* dan algoritma A*, selanjutnya dilakukan pencarian rute minimum dengan cara menghitung semua kemungkinan rute pada data sampel.

Keterangan selengkapnya mengenai tahap pelaksanaan uji coba bisa dilihat pada Lampiran 1.

4.2.1 Hasil Pengujian

Untuk mengetahui keoptimalan hasil perhitungan metode *heuristic assignment* Fisher-Jaikumar dan Algoritma A*, berikut ini diberikan tabel perbandingannya dengan hasil perhitungan pada semua kemungkinan jalur untuk data sampel.

Tabel 4.1 Perbandingan Hasil Perhitungan

Uji Ke	Jumlah Titik Rawan	Total Jarak		Waktu	
		Perhitungan dengan Metode Heuristic Assignment dan Algoritma A* (piksel)	Perhitungan Pada Semua Kemungkinan Jalur (piksel)	Perhitungan dengan Metode Heuristic Assignment dan Algoritma A* (milidetik)	Perhitungan Pada Semua Kemungkinan Jalur (milidetik)
1	7	2327	1624	0	15
2	8	1221	1149	15	31
3	9	2178	1599	16	281
4	10	1409	1365	14	5905
5	11	2623	2318	16	151093

Dari hasil pengujian pada Tabel 4.1, bisa diketahui bahwa selisih jarak yang dihasilkan oleh metode *heuristic assignment* Fisher-Jaikumar dan algoritma A* dengan jarak minimum sebenarnya bervariasi. Berikut ini diberikan nilai galat untuk masing-masing pengujian pada Tabel 4.2.

Tabel 4.2 Hasil Perhitungan Galat

Uji Ke	Galat (%)
1	43,29
2	6,27
3	36,21
4	3,22
5	13,16

Dari Tabel 4.1 dan Tabel 4.2 bisa diketahui bahwa selisih paling kecil ada pada pengujian keempat, yaitu menghasilkan selisih jarak sebesar 44 dengan nilai galat 3,22%. Sedangkan nilai galat terbesar terdapat pada pengujian pertama yaitu sebesar 43,29%. Selisih yang besar ini kemungkinan disebabkan pada proses optimasi dengan algoritma A* digunakan fungsi *heuristic*. Nilai dalam fungsi ini

diperoleh dari hasil perkiraan sehingga angka yang dihasilkan juga kurang akurat.

Untuk waktu pelaksanaan proses, metode *heuristic assignment* Fisher-Jaikumar dan algoritma A* hanya mengalami penambahan waktu yang sedikit pada tiap penambahan jumlah titik rawan. Sedangkan pencarian rute dengan menghitung semua kemungkinan yang ada memerlukan waktu hampir sama dengan metode *heuristic assignment* Fisher-Jaikumar dan algoritma A* pada saat jumlah titik rawan kurang dari delapan. Namun untuk jumlah titik rawan lebih dari delapan, waktu yang dibutuhkan untuk melakukan proses meningkat drastis. Begitu juga ketika ditambahkan menjadi sebelas titik rawan. Hal ini disebabkan penambahan jumlah kemungkinan rute yang harus dihitung akan semakin besar tiap kali dilakukan penambahan jumlah titik rawan. Penambahan jumlah kemungkinan rute yang harus dihitung di tunjukkan pada persamaan 4.1.

```
Jumlah kemungkinan rute = ((n-1)!) x Jumlah Kombinasi
```

(4.1)

Di mana n adalah jumlah titik rawan sedangkan yang dimaksud jumlah kombinasi adalah jumlah kemungkinan pengelompokan titik-titik rawan ke dalam sejumlah mobil patroli. Misalkan untuk titik rawan berjumlah tujuh dengan dua buah mobil patroli, memiliki jumlah kombinasi sebanyak tiga yaitu kombinasi pertama dua titik rawan untuk mobil pertama dan enam titik rawan untuk mobil kedua. Kombinasi kedua terdiri dari tiga titik rawan untuk mobil pertama dan lima titik rawan untuk mobil kedua. Dan kombinasi yang terakhir terdiri dari empat titik rawan untuk mobil pertama dan empat titik rawan juga untuk mobil kedua. Dengan asumsi titik rawan (1) adalah kantor polisi sehingga digunakan sebagai *start node* dan *goal node* untuk semua mobil patroli.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil penelitian dalam tugas akhir ini bisa diperoleh kesimpulan sebagai berikut:

1. Dari hasil pengujian, pencarian rute menggunakan metode *heuristic assignment* Fisher-Jaikumar dan Algoritma A* dihasilkan nilai jarak yang mendekati optimum dengan tingkat kesalahan rata-rata sebesar 20,43%.
2. Waktu yang dibutuhkan untuk menyelesaikan suatu pencarian rute dengan metode *heuristic assignment* Fisher-Jaikumar dan algoritma A* lebih cepat daripada menghitung semua kemungkinan rute.
3. Penggunaan metode *heuristic assignment* Fisher-Jaikumar dan algoritma A* lebih cocok untuk jumlah data yang besar. Sedangkan untuk data yang berukuran kecil lebih baik dengan menghitung semua kemungkinan rute yang ada karena hasil yang diperoleh pasti optimal.

5.2 Saran

Saran yang dapat diberikan setelah selesainya penelitian ini yaitu sistem pencarian rute menggunakan metode *heuristic assignment* Fisher-Jaikumar dan Algoritma A* dapat dikembangkan sehingga mencakup patroli keamanan yang dilaksanakan oleh semua unit patroli yang ada di dalam kota baik itu dari pihak Polresta maupun Polsekta. Pengembangan ini meliputi titik keberangkatan patroli keamanan yang berbeda-beda, bobot masing-masing titik rawan (tingkat kerawanan pada titik rawan tersebut dibandingkan dengan titik-titik rawan lainnya) yang mempengaruhi frekuensi kunjungan patroli keamanan serta pembagian jadwal patroli untuk masing-masing unit patroli dari Polresta maupun Polsekta. Dengan demikian sistem patroli keamanan ini bisa lebih efektif dan efisien.

DAFTAR PUSTAKA

Aho, Alfred V et al. 1974. *The Design and Analysis of Computer Algorithm*. America: Addison-Wesley Publishing Company.

Alfandari, Laurent et al. 2001. *A Two-Phase Path-Relinking Algorithm for the Generalized Assignment Problem.* 4 th Metaheuristic International Conference (MIC). Porto, Portugal.

Anonymous. 2005. *Assignment problem*, diakses pada tanggal 6 Maret 2007 dari http://en.wikipedia.org/wiki/assignment_problem.htm.

Anonymous. *A* search algorithm*, diakses pada tanggal 3 Pebruari 2007 dari http://en.wikipedia.org/wiki/A-star_algorithm.htm

Anonymous. *Algoritma dan Flowchart*, diakses pada tanggal 10 Maret 2007 dari http://www.faculty.petra.ac.id/thiang/download/dkp/Algoritma dan Flowchart.doc.

Anonymous. *Hamiltonian path,* diakses pada tanggal 6 Maret 2007 dari http://en.wikipedia.org/wiki/hamiltonian_path.htm.

Anonymous, *Travelling Salesman Problem*, diakses pada tanggal 3 Pebruari 2007 dari http://en.wikipedia.org/wiki/Traveling_Salesman_Problem.htm.

Golin, Mordecai. 2003. *Design and Analysis of Algorithms*. Department of Computer Science, HKUST.

Koskosidis, Yiannis A & Powell, Warren B. 1992. Clustering Algorithms for Consolidation of Customer Orders into Vehicle Shipments. *Transpn. Res. –B,* 26B(5): 365-379.

Kusumadewi, Sri. 2003. *Artificial Intelligence (teknik dan Aplikasinya)*. Yogyakarta: Graha Ilmu.

Ladjamudin, Al-Bahra. 2004. *Konsep Sistem Basis Data dan Implementasinya*. Yogyakarta: Graha Ilmu.

Lester, Patrick. 2005. *A* Pathfinding for Beginners*. Diakses pada tanggal 4 Pebruari 2007 dari http://www.policyalmanac.org/games/aStarTutorial.htm.

Munir, Rinaldi. 2003. *Metode Numerik*. Bandung: Informatika Bandung.

Munir, Rinaldi. 2004. *1. Algoritma Brute Force (lanjutan) 2. Heuristik*. Bahan Kuliah-2 Strategi Algoritmik. Departemen Teknik Informatika, Institut Teknologi Bandung, Bandung.

Robi'in, Bambang. 2005. *Manajemen dan Administrasi Database dengan SQL Server 2000*. Yogyakarta: Andi Yogyakarta.

Setiyono, Budi. 2002. Pembuatan Perangkat Lunak Penyelesaian Multi Travelling Salesman Problem (m-TSP). *KAPPA*, 3(2): 55-65.

Siang, Jong Jek. 2002. *Matematika Diskrit dan Aplikasinya pada Ilmu Komputer*. Yogyakarta: Andi.

Lampiran 1

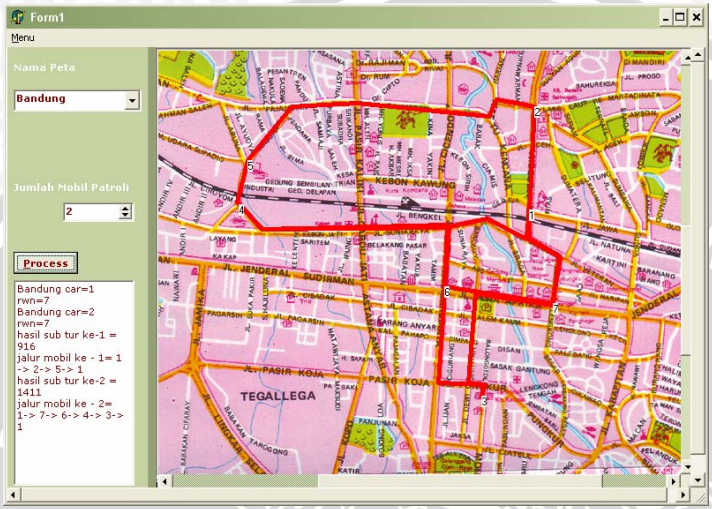
Pengujian Data Sampel

Berikut ini adalah proses pengujian data sampel. Pengujian dilakukan sebanyak lima kali. Data dari masing-masing pengujian bisa dilihat pada Lampiran 2 dan Lampiran 3.

Titik Rawan Sampel ke-1

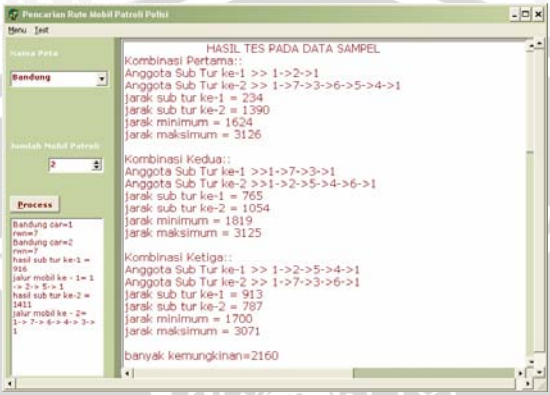
Selanjutnya dilakukan pencarian rute menggunakan metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A*.

Pencarian Rute dengan metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A* pada Sampel ke-1



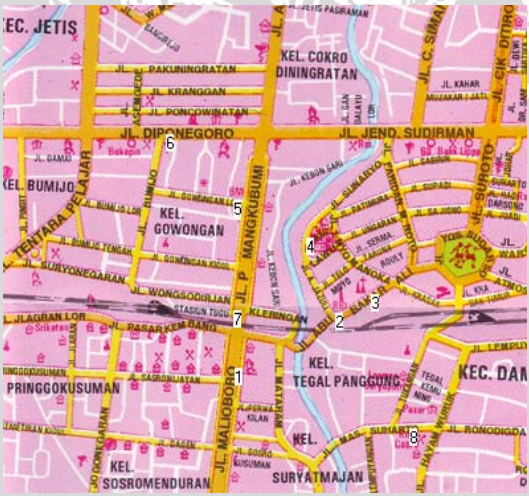
Setelah diketahui hasil pencarian rute menggunakan metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A*, berikutnya dilakukan pengujian data sampel dengan melakukan perhitungan pada semua kemungkinan rute yang ada.Untuk memudahkan perhitungan, maka seluruh kemungkinan jalur dibagi menjadi tiga kelompok. Masing-masing kelompok memiliki ciri kombinasi tersendiri dan disebut sebagai kombinasi pertama, kedua dan ketiga. Untuk kombinasi pertama memiliki anggota sub tur pertama sebanyak dua titik rawan dan anggota sub tur kedua sebanyak enam titik rawan. Kombinasi kedua yaitu terdiri dari sub tur pertama dengan anggota sebanyak tiga titik rawan dan sub tur kedua dengan anggota lima titik rawan. Yang terakhir yaitu kombinasi ketiga, sub tur pertama memiliki anggota empat titik rawan dan sub tur kedua juga memiliki anggota empat titik rawan. Untuk semua kombinasi pada semua kasus berlaku titik rawan (1) menjadi anggota semua sub tur, yaitu sebagai *start node* dan *goal node*.

Pencarian Rute dengan Perhitungan Semua Kemungkinan pada Sampel ke-1



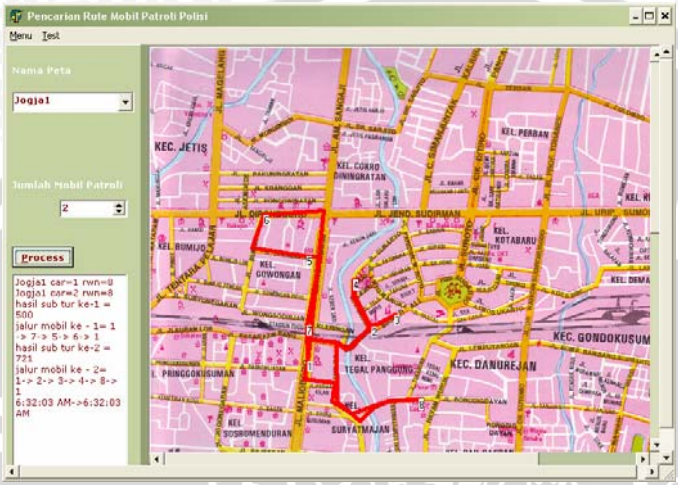
Selanjutnya dilakukan pengujian pada data sampel kedua. Berikut ini posisi titik rawan pada Data Sampel kedua.

Titik Rawan Sampel ke-2



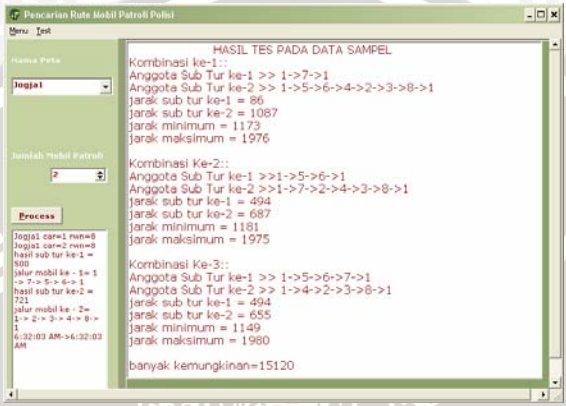
Kemudian dilakukan pencarian rute pada data sampel ke-2 menggunakan metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A*.

Pencarian Rute dengan metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A* pada Sampel ke-2



Setelah itu dilakukan perhitungan semua kemungkinan rute dengan cara yang sama seperti sebelumnya, yaitu dibagi menjadi tiga kelompok kombinasi. Namun kombinasi pada pencarian rute ini memiliki jumlah anggota yang berbeda dari data sampel sebelumnya. Kombinasi pertama terdiri dari sub tur pertama dengan anggota dua titik rawan dan sub tur kedua dengan anggota tujuh titik rawan. Kombinasi kedua terdiri dari sub tur pertama dengan tiga titik rawan dan sub tur kedua dengan enam titik rawan. Dan yang terkhir kombinasi ketiga terdiri dari empat titik rawan pada sub tur pertama dan lima titik rawan pada sub tur kedua.

Pencarian Rute dengan Perhitungan Semua Kemungkinan pada Data Sampel ke-2

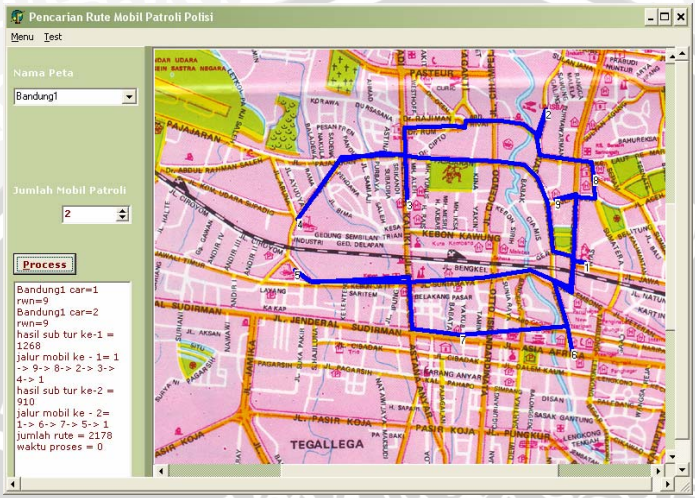


Berikutnya dilakukan pengujian pada sampel ke-3.

Titik Rawan Sampel ke-3

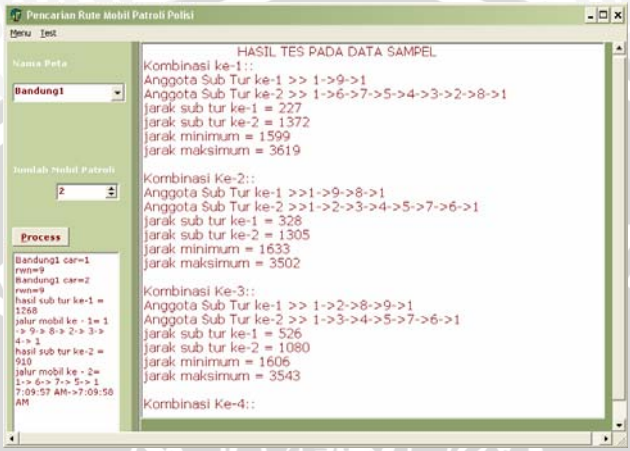
Kemudian dilakukan pencarian rute menggunakan metode *heuristic assignment* Fisher-Jaikumar dan algoritma A*.

Pencarian Rute dengan metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A* pada Sampel ke-3

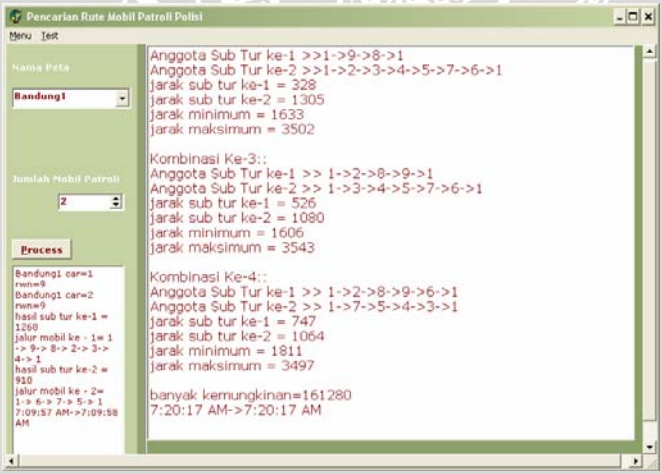


Hasil tersebut kemudian dibandingkan dengan hasil perhitungan semua kemungkinan. Untuk data sampel ketiga, jumlah titik rawan adalah sembilan, sehingga untuk mempermudah perhitungan semua kemungkinannya, titik rawan dibagi menjadi empat kombinasi. Kombinasi pertama terdiri dari sub tur pertama dengan anggota dua titik rawan dan sub tur kedua dengan anggota delapan titik rawan. Kombinasi kedua terdiri dari tiga titik rawan sebagai anggota sub tur pertama dan tujuh titik rawan untuk anggota sub tur kedua. Kombinasi berikutnya yaitu kombinasi ketiga terdiri dari sub tur pertama dengan anggota empat titik rawan, dan sub tur kedua dengan anggota enam titik rawan. Dan yang terakhir yaitu kombinasi keempat terdiri dari sub tur pertama dengan anggota lima titik rawan dan sub tur kedua dengan anggota juga lima titik rawan. Hasil perhitungan pada semua kemungkinan ditunjukkan pada Gambar Pencarian Rute dengan Perhitungan Semua Kemungkinan pada Data Sampel ke-3 dan Gambar Lanjutan Pencarian Rute dengan Perhitungan Semua Kemungkinan pada Data Sampel ke-3.

Pencarian Rute dengan Perhitungan Semua Kemungkinan pada Sampel ke-3



Lanjutan Pencarian Rute dengan Perhitungan Semua Kemungkinan pada Sampel ke-3

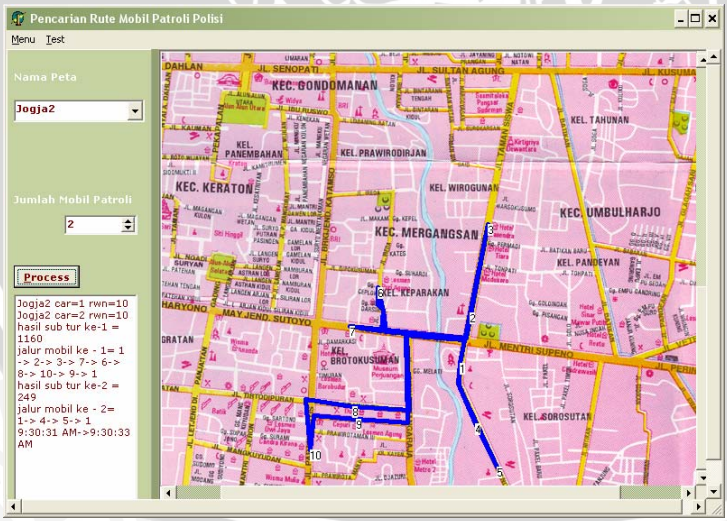


Selanjutnya dilakukan pengujian pada data sampel keempat. Posisi titik-titik rawan pada data sampel keempat seperti ditunjukkan dalam Gambar Titik Rawan Data Sampel ke-4.

Titik Rawan Sampel ke-4

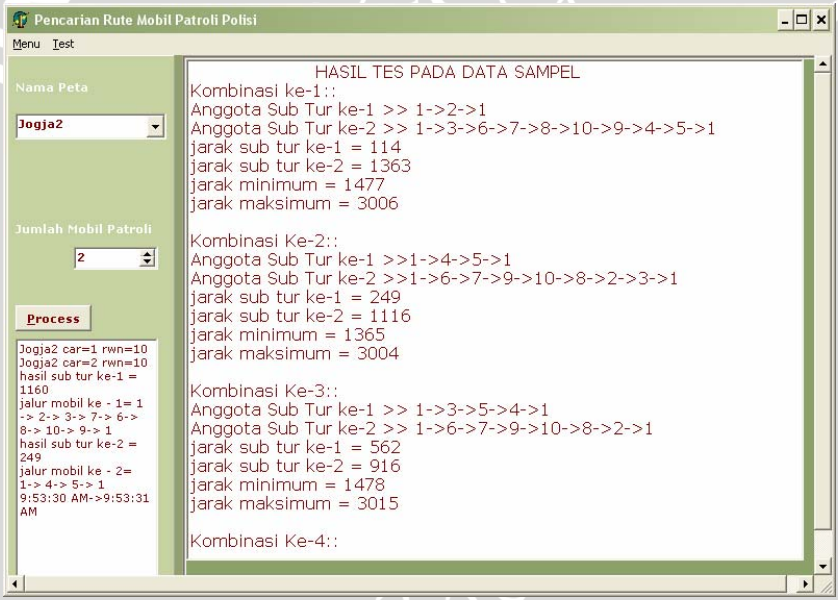
Dari titik-titik rawan tersebut kemudian dilakukan pencarian rute menggunakan metode *heuristic assignment* dan algoritma A*.

Pencarian Rute dengan metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A* pada Sampel ke-4

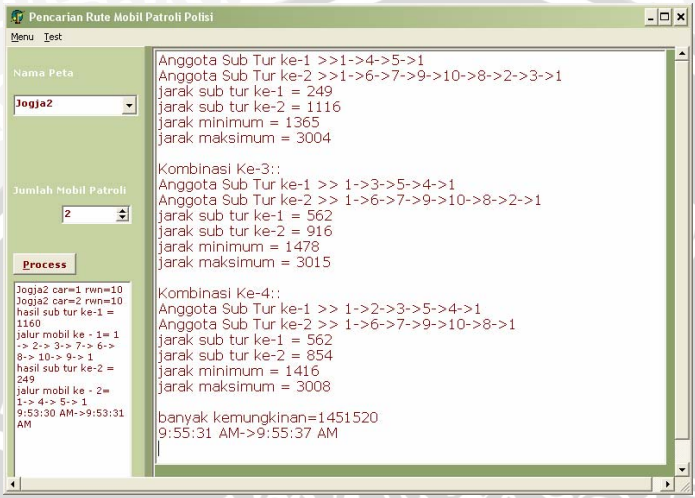


Untuk pencarian rute dengan perhitungan semua kemungkinan, sepuluh titik rawan pada sampel keempat dibagi menjadi empat kombinasi. Kombinasi pertama terdiri dari sub tur pertama dengan anggota dua titik rawan dan sub tur kedua sembilan titik rawan. Kombinasi kedua terdiri dari sub tur pertama yang memiliki anggota tiga titik rawan dan sub tur kedua memiliki anggota delapan titik rawan. Kombinasi ketiga memiliki empat titik rawan pada sub tur pertama dan tujuh titik rawan pada sub tur kedua. Kombinasi terakhir terdiri dari sub tur pertama dengan anggota lima titik rawan dan sub tur kedua memiliki anggota enam titik rawan.

Pencarian Rute dengan Perhitungan Semua Kemungkinan pada Sampel ke-4



Lanjutan Pencarian Rute dengan Perhitungan Semua Kemungkinan pada Data Sampel ke-4

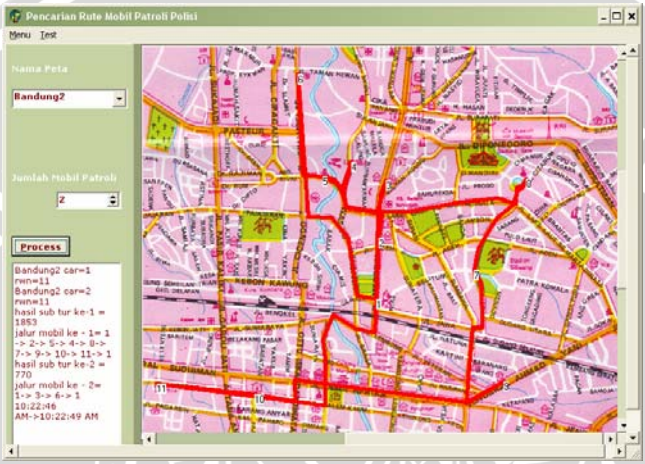


Berikutnya adalah pengujian sampel kelima.

Titik Rawan Sampel Kelima

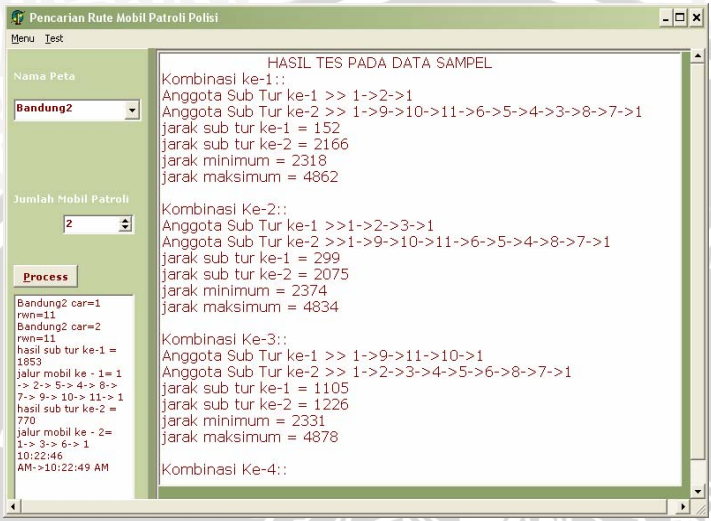
Selanjutnya dilakukan pencarian rute dengan metode *heuristic assignment* dan algoritma A*.

Pencarian Rute dengan metode *Heuristic Assignment* Fisher-Jaikumar dan Algoritma A* pada Sampel ke-5

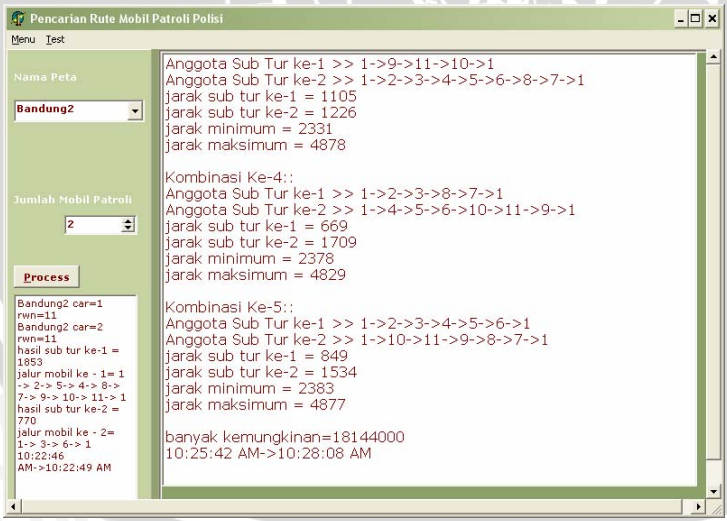


Karena jumlah titik rawan dalam sampel kelima adalah sebelas titik rawan, maka untuk perhitungan semua kemungkinan rute, dibagi menjadi lima kombinasi. Kombinasi pertama terdiri dari dua titik rawan anggota sub tur pertama dan sepuluh titik rawan anggota sub tur kedua. Kombinasi kedua terdiri dari tiga anggota sub tur pertama dan sembilan titik rawan anggota sub tur kedua. Selanjutnya kombinasi ketiga terdiri dari sub tur pertama dengan anggota empat titik rawan dan sub tur kedua dengan anggota delapan titik rawan. Kombinasi keempat terdiri dari sub tur pertama yang memiliki lima titik rawan dan sub tur kedua memiliki anggota tujuh titik rawan. Dan yang terakhir, pada kombinasi kelima terdiri dari sub tur pertama dengan anggota enam titik rawan serta sub tur kedua dengan anggota enam titik rawan juga.

Pencarian Rute dengan Perhitungan Semua Kemungkinan pada Data Sampel ke-5



Lanjutan Pencarian Rute dengan Perhitungan Semua Kemungkinan pada Data Sampel ke-5



Lampiran 2

Tabel Koordinat Titik Rawan

Uji Coba Ke	Titik Rawan	x	y
1	1	721	768
	2	727	651
	3	667	979
	4	392	762
	5	402	712
	6	625	855
	7	748	874
2	1	186	372
	2	262	331
	3	289	317
	4	240	275
	5	185	247
	6	134	198
	7	185	329
	8	318	417
3	1	735	768
	2	690	591
	3	529	696
	4	402	718
	5	399	777
	6	720	869
	7	591	852
	8	745	667
	9	701	695
4	1	394	916
	2	405	860
	3	425	762
	4	411	983
	5	435	1031
	6	303	832

Uji Coba Ke	Titik Rawan	x	y
	7	272	875
	8	275	964
	9	279	978
	10	228	1012
5	1	723	758
	2	726	682
	3	735	610
	4	692	587
	5	656	605
	6	625	478
	7	845	722
	8	910	608
	9	882	860
	10	571	876
	11	449	863

Lampiran 3

Tabel Jarak Antar Titik Rawan

Uji Coba Ke-1							
Titik Rawan	1	2	3	4	5	6	7
1	0	117	351	377	431	196	128
2	117	0	484	486	417	311	241
3	374	487	0	491	555	544	287
4	379	493	438	0	51	318	420
5	422	368	542	51	0	371	470
6	195	311	177	322	376	0	124
7	127	246	293	426	482	124	0

Uji Coba Ke-2								
Titik Rawan	1	2	3	4	5	6	7	8
1	0	109	148	177	134	251	43	190
2	116	0	30	71	196	309	102	236
3	150	30	0	96	227	345	133	192
4	171	63	91	0	257	360	164	299
5	136	192	235	251	0	147	99	334
6	249	307	339	376	107	0	211	441
7	43	107	135	163	101	208	0	236
8	198	231	193	294	339	440	234	0

Uji Coba Ke-3									
Titik Rawan	1	2	3	4	5	6	7	8	9
1	0	222	272	450	371	155	286	139	111
2	207	0	287	390	484	289	422	130	168
3	266	284	0	214	275	338	207	311	252
4	447	386	215	0	64	445	305	409	360
5	385	451	272	63	0	385	252	471	417
6	157	289	337	434	390	0	130	263	177
7	262	387	206	309	251	130	0	390	303
8	168	129	318	420	478	261	383	0	73
9	116	123	251	359	408	185	306	79	0

Uji Coba Ke-4										
Titik Rawan	1	2	3	4	5	6	7	8	9	10
1	0	57	157	70	124	170	154	232	245	335
2	57	0	100	127	181	167	159	237	266	340
3	157	100	0	228	281	272	261	344	353	434
4	71	127	228	0	54	233	228	306	307	405
5	125	181	281	54	0	288	278	358	381	464
6	168	188	277	242	291	0	85	227	242	273
7	161	160	258	226	280	83	0	218	224	180
8	240	234	349	308	355	221	167	0	134	99
9	255	252	347	315	363	234	189	143	0	85
10	336	336	441	421	452	260	183	109	83	0

Uji Coba Ke-5											
Titik Ra-wan	1	2	3	4	5	6	7	8	9	10	11
1	0	76	150	211	199	336	129	313	221	286	426
2	76	0	74	136	128	291	140	240	301	346	483
3	149	73	0	144	138	288	200	252	373	434	551
4	210	137	148	0	72	229	263	314	433	528	531
5	205	128	143	66	0	152	321	310	411	371	490
6	360	288	285	220	150	0	404	449	594	469	581
7	223	195	324	360	290	432	0	135	231	473	599
8	314	237	256	309	307	449	137	0	354	597	723
9	225	299	409	435	488	666	235	374	0	346	474
10	319	363	440	426	361	465	491	617	349	0	123
11	458	479	544	548	487	595	623	711	471	123	0