UNIVERSITAS INDONESIA

PENGGUNAAN ALGORITMA GENETIKA DALAM MASALAH JALUR TERPENDEK PADA JARINGAN DATA

SKRIPSI

RAMA M SUKATON
0606067742

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
PROGRAM STUDI SARJANA MATEMATIKA
DEPOK
JUNI 2011

UNIVERSITAS INDONESIA

PENGGUNAAN ALGORITMA GENETIKA DALAM MASALAH JALUR TERPENDEK PADA JARINGAN DATA

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar sarjana sains

RAMA M SUKATON
0606067742

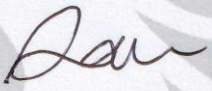
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
PROGRAM STUDI SARJANA MATEMATIKA
DEPOK
JUNI 2011

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya sendiri,

dan semua sumber baik yang dikutip maupun dirujuk

telah saya nyatakan dengan benar.

Nama	: Rama M. Sukaton
NPM	: 0606067742
Tanda Tangan	: 
Tanggal	: 14 Juni 2011

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh	:	
Nama	:	Rama M. Sukaton
NPM	:	0606067742
Program Studi	:	Sarjana Matematika
Judul Skripsi	:	Penggunaan Algoritma Genetika Dalam Masalah Jalur Terpendek Pada Jaringan Data

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Sains pada Program Studi S1 Matematika, Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Indonesia

DEWAN PENGUJI

Pembimbing	: Dr. Yudi Satria, M.T.	( )
Pembimbing	: Rahmi Rusin, M.ScTech.	( )
Penguji	: Dr. Alhadi Bustamam, M.Kom.	( )
Penguji	: Dr.rer.nat. Hendri Murfi, M.Kom.	( )
Penguji	: Drs. Suryadi MT, M.T.	( )

Ditetapkan di	:	Depok
Tanggal	:	14 Juni 2011

KATA PENGANTAR

Puji syukur atas kehadirat Tuhan Yang Maha Esa atas kasih, rahmat, dan karunia-Nya sehingga penulis dapat mengerjakan dan menyelesaikan penulisan skripsi ini. Ucapan terima kasih penulis sampaikan kepada:

1. Kedua orang tua penulis yang selalu memberikan do'a dan dukungan.
2. Dr. Yudi Satria, M.T. dan Rahmi Rusin, M.ScTech. selaku pembimbing skripsi yang telah dengan sangat sabar meluangkan waktu dan pikiran serta memberikan arahan, motivasi, bimbingan kepada penulis dalam mengerjakan skripsi ini.
3. Dr. Alhadi Bustamam, M.Kom., Dr.rer.nat. Hendri Murfi, M.Kom., dan Drs. Suryadi MT, M.T. selaku dewan penguji seminar kolokium yang telah memberikan kritik dan saran demi kesempurnaan skripsi ini.
4. Seluruh dosen dan staff Departemen Matematika FMIPA UI, khususnya Mba Santhi yang sangat banyak membantu dan Dr. Kiki Ariyanti Sugeng selaku pembimbing akademik penulis.
5. Dra. Sri Harini, M.Kom. yang telah memberikan banyak masukan kepada penulis dan Dr. Hengki Tasman.
6. Mas Bara selaku kakak yang telah memberikan dukungan kepada penulis.
7. Teman-teman mahasiswa Matematika UI, khususnya angkatan 2006: Sutisna, Indah, Oppie, Teguh, Dodi, Rafly, Ali, Pangky, Anggha, Michael, Oza, Aliman, Ichwan, Syafira, Yuri, Cims, Rendi, Rian, Billy, Oppie, Mei, Inne, Rizkyatul, Rahanti, Stefani, Alfa, Mella, Milla, Tami, Annisa, Widya, Latief, Hot, Farah, Putri Helmet, Purwita, Nurgi, Lena, Nadia, Reza, Rifza, Yunita, Tika, Rontu, Lani, Dita, Budi, Rita, Indra, Puspa, Poe, Tasya, Billy, Rahmanita, Kiki, Nobo, Rita, Lee.

Akhir kata, penulis mohon maaf atas segala kesalahan atau kekurangan. Semoga skripsi ini dapat bermanfaat bagi pembaca pada umumnya dan khususnya bagi penulis.

Penulis

2011

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama	: Rama M. Sukaton
NPM	: 0606067742
Program Studi	: Sarjana Matematika
Departemen	: Matematika
Fakultas	: Matematika dan Ilmu Pengetahuan Alam
Jenis karya	: Skripsi

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty Free Right*) atas karya ilmiah saya yang berjudul :

Penggunaan Algoritma Genetika Dalam Masalah Jalur Terpendek Pada Jaringan Data

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/format-kan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan mempublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok

Pada tanggal : 14 Juni 2011

Yang menyatakan

(Rama M. Sukaton)

ABSTRAK

Nama : Rama M. Sukaton
Program Studi : Matematika
Judul : Penggunaan Algoritma Genetika Dalam Masalah Jalur Terpendek Pada Jaringan Data

Dalam teori graf, masalah jalur terpendek merupakan suatu masalah pencarian jalur antara dua verteks sedemikian sehingga jumlah bobot dari busur penyusunnya adalah minimum. Masalah jalur terpendek ini salah satunya dapat ditemui pada jaringan data, yakni proses *routing* pada saat pengiriman data dari *node* sumber ke *node* tujuan. Terdapat beberapa algoritma atau metode yang dapat memecahkan masalah jalur terpendek ini, pada skripsi ini akan dibahas penerapan algoritma genetika yang didasarkan prinsip evolusi biologi dalam penyelesaian jalur terpendek. Operator dasar yang digunakan pada skripsi ini adalah roda *roulette* untuk reproduksi, *order crossover* untuk *crossover*, dan *insertion mutation* untuk mutasi. Kinerja algoritma genetika akan diuji dengan menggunakan data dari *OR-Library*. Berdasarkan hasil percobaan diperoleh bahwa algoritma genetika cukup baik untuk digunakan dalam penyelesaian masalah jalur terpendek. Selain itu, ditunjukkan bahwa perubahan nilai parameter algoritma genetika ternyata mempengaruhi kinerja algoritma genetika dalam memperoleh solusi.

Kata Kunci : Algoritma Genetika, Masalah Jalur Terpendek, *Routing*, Jaringan Data
iv+55 halaman : 2 lampiran
Daftar Pustaka : 19 (1989 – 2011)

ABSTRACT

Name : Rama M. Sukaton
Study Program : Mathematics
Title : The Use of Genetic Algorithm in Shortest Path Problem on Data Network

In graph theory, shortest path problem is a problem of finding a path between two vertices such that the total cost of the constituent edges is minimum. Shortest path problem can be found in data networks, namely routing process, when transmitting data from a source node to a destination node. There are several algorithms or methods that can solve this problem. In this final project, genetic algorithm based on principles of evolutionary biology is used to solve it. The basic operator for the genetic algorithm that used are the roulette-wheel for reproduction, order crossover, and insertion mutation. The performance of the genetic algorithm will be applied by using data from OR-Library. Based on the experiment result, the genetic algorithm is good enough to solve the shortest path problem. In addition, changes in values of parameters will affect the performance of the genetic algorithm in obtaining a solution.

Keywords : Genetic Algorithm, Shortest Path Problem, Routing, Data Network
iv+55 pages : 2 attachments
Bibliography : 19 (1989 – 2011)

DAFTAR ISI

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

DAFTAR LAMPIRAN

Halaman

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Perkembangan yang pesat dalam jaringan internet, mengakibatkan permintaan lalu lintas untuk jaringan internet baik oleh berbagai komunitas bisnis maupun individu, telah mendekati dua kali lipat tiap tahunnya (K.G. Coman dan A.M. Odlyzko, 2001). Bahkan dari tahun 2000 hingga 2011 perkembangan internet di dunia mencapai 480,4% dengan 2.095.006.005 pengguna dari data terakhir yang diperoleh (*Internet World Stats*, 2011). Penyedia layanan sambungan internet yang lebih dikenal dengan ISPs (*Internet Service Providers*) mencoba memenuhi peningkatan permintaan lalu lintas tersebut dengan teknologi baru dan meningkatkan pemanfaatan dari sumber daya yang ada. *Routing* paket data dalam hal ini dapat mempengaruhi kemampuan atau utilisasi sumber daya jaringan (R. Kumar dan M. Kumar, 2010).

Routing merupakan suatu proses pencarian jalur dalam suatu jaringan dari *node* sumber ke *node* tujuan yang digunakan sebagai jalur data. *Routing* memiliki peran fundamental dalam internet. *Routing* merupakan fungsi yang bertanggung jawab membawa data melewati sekumpulan jalur dalam jaringan dengan cara memilih jalur terbaik untuk dilewati data. Tugas *routing* tersebut dilakukan oleh perangkat jaringan yang disebut sebagai *router*. Dalam tugasnya *router* menggunakan yang dinamakan protokol *routing*, untuk menentukan jalur terbaik. Protokol *routing* mengijinkan *router-router* untuk berbagi informasi tentang jaringan dan koneksi antar *router*. Dalam *intra-domain* Internet Routing Protocol (IRP), OSPF (*Open Shortest Path First*) adalah protokol *routing* yang sering digunakan (R. Kumar dan M. Kumar, 2010).

Dalam jaringan dinamis dan sangat besar, *routing* menjadi sangat kompleks karena banyak potensi dalam pertengahan perjalanan suatu paket dapat terhalang sebelum mencapai tujuannya. Selain itu, pengguna pun dapat masuk dan keluar dari topologi logika jaringan. Sehingga dibutuhkan algoritma *routing* yang

baik dan mampu menekan waktu dalam *update* jaringan ataupun jika terjadi kesalahan dalam jaringan.

Permasalahan *routing* ini dapat direpresentasikan sebagai masalah jalur terpendek untuk memudahkan penyelesaiannya. Masalah jalur terpendek adalah masalah menemukan suatu jalur antara dua simpul sedemikian sehingga jumlah bobot dari busur penyusunnya dapat seminimal mungkin (R. Kumar dan M. Kumar, 2010). Beberapa metode algoritma yang telah dikembangkan untuk menyelesaikan persoalan jalur terpendek diantaranya algoritma Djikstra, algortima Floyd-Warshall, dan algoritma Bellman-Ford. Sementara metode yang paling efisien untuk permasalahan jalur terpendek dalam jaringan data adalah algoritma Djikstra (R. Kumar dan M. Kumar, 2010). Protokol *routing* OSPF (*Open Shortest Path First*) menggunakan algoritma *link-state* juga dikenal dengan algoritma Djikstra atau algoritma *Shortest Path First* (SPF) dalam penentuan jalur terpendek ini. Namun, pada jaringan dinamis yang sangat besar, algoritma Djikstra menjadi tidak efisien karena simpul-simpul pada jaringan akan dikunjungi kembali sehingga banyak komputasi atau perhitungan-perhitungan yang diulang (R. Kumar dan M. Kumar, 2010).

Pada skripsi ini, pendekatan yang digunakan untuk menyelesaikan masalah jalur terpendek dalam jaringan data tersebut adalah algoritma genetika. Algoritma genetika adalah metode *adaptive* yang biasa digunakan untuk memecahkan suatu pencarian nilai dalam masalah optimasi. Peletak prinsip dasar sekaligus pencipta algoritma genetika adalah John Holland yang dipublikasikan pada tahun 1975 (M. Obitko, 1998). Dengan meniru teori evolusi, algoritma genetika dapat digunakan untuk mencari solusi permasalahan-permasalahan dalam dunia nyata. Sebelum algoritma genetika dapat dijalankan, maka sebuah kode yang sesuai untuk persoalan harus dirancang. Untuk ini maka solusi layak dalam ruang permasalahan dikodekan dalam bentuk kromosom yang terdiri atas komponen genetik terkecil yaitu gen. Dengan teori evolusi dan teori genetika, di dalam penerapan algoritma genetika akan melibatkan beberapa operator, yaitu reproduksi, *crossover*, dan mutasi. Adapun pemilihan algoritma genetika pada skripsi ini lebih ditekankan bahwa algoritma genetika dapat digunakan sebagai alternatif lain untuk penyelesaian masalah jalur terpendek.

Jenis dari operator dasar algoritma genetika bermacam-macam dan terus berkembang. Pada skripsi ini, operator genetik yang akan digunakan adalah roda *roulette selection* untuk reproduksi, *Order Crossover* (OX) untuk *crossover*, dan *insertion mutation* untuk mutasi. Secara umum, tahapan dari algoritma genetika diawali dengan pembentukan populasi awal. Selanjutnya dilakukan proses yang menggunakan ketiga operator genetik tersebut untuk membentuk populasi baru yang akan digunakan untuk generasi berikutnya. Banyaknya proses *crossover* dan mutasi bergantung pada masing-masing nilai parameter probabilitas yang telah ditentukan sebelumnya. Proses atau tahapan pada algoritma genetik tersebut dilakukan berulang kali sampai mencapai suatu kriteria berhenti, dalam hal ini batas generasi yang telah ditentukan.

1.2 Perumusan Masalah

Bagaimana algoritma genetika dapat digunakan untuk menyelesaikan masalah jalur terpendek dalam jaringan data.

1.3 Tujuan Penulisan

Penulisan skripsi ini bertujuan untuk menerapkan algoritma genetika dalam menyelesaikan masalah jalur terpendek dalam suatu jaringan data. Selain itu juga dilakukan implementasi dan simulasi algoritma genetika dengan menggunakan bantuan perangkat lunak.

1.4 Pembatasan Masalah

Menggunakan data pada *OR-Library* (*OR-Library*, 2011) untuk melakukan percobaan yang dapat dilihat pada akhir Bab 4 skripsi ini.

1.5 Sistematika Penulisan

Skripsi ini dibagi dalam lima bab yakni Bab 1 berisi pendahuluan, Bab 2 berisi landasan teori yang membahas konsep dasar jaringan *routing* (*network routing*), teori graf, dan algoritma genetika. Bab 3 membahas penyelesaian jalur terpendek dalam suatu jaringan dengan menggunakan algoritma genetika. Bab 4 membahas simulasi dan implementasi algoritma genetika dalam menyelesaikan masalah jalur terpendek dengan menggunakan bantuan Matlab serta analisa beberapa hasil percobaan. Bab 5 berisi penutup dari sisi uraian skripsi yang berisi kesimpulan dan saran.

BAB 2
LANDASAN TEORI

2.1 *Routing*

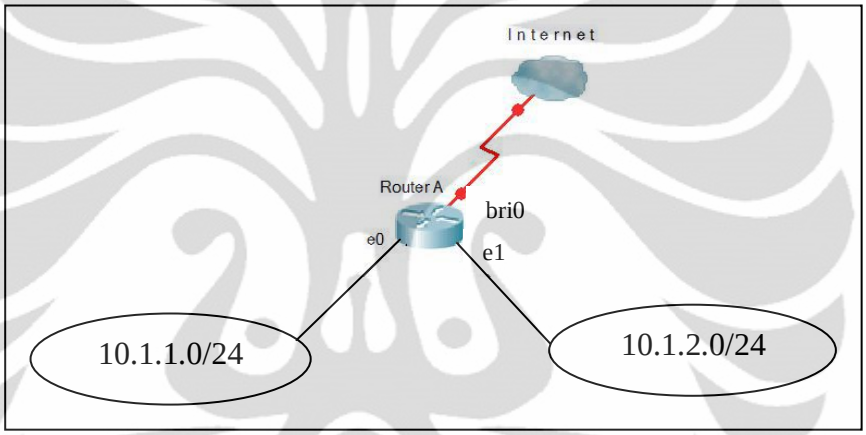
Routing adalah suatu proses menemukan jalur antara simpul dalam suatu jaringan komputer. Terdapat dua tipe kebijakan *routing* yang utama, yaitu *static routing* dan *dynamic routing*. Dalam *static routing*, jalur antara simpul-simpul di tentukan secara manual berdasarkan faktor tertentu dan disimpan dalam tabel *routing* (R. Kumar dan M. Kumar, 2010). Semua paket data antara dua simpul mengikuti jalur yang sama. Namun ketika topologi jaringan berubah disebabkan faktor tertentu seperti salah satu simpul dalam jalur dalam keadaan tidak berfungsi (*down*), maka *static routing* ini gagal mengantar paket data sampai tujuannya. Sementara *dynamic routing* dapat memperbaharui informasi topologi jaringan dalam interval waktu tertentu dan juga secara otomatis dapat mengubah jalur-jalur dalam tabel *routing* jika terjadi perubahan dalam topologi jaringan.

Proses *routing* itu sendiri dilakukan oleh perangkat jaringan yang disebut *router* dan diatur oleh suatu protokol. *Router* ini dapat berupa komputer atau perangkat jaringan khusus tertentu. Dalam melakukan tugas *routing* tersebut, *router* harus mengetahui informasi-informasi seperti (R. A. F. Adriansyah, 2008):

1. Banyaknya *port* yang dimiliki dan tipe-tipenya untuk mendukung beberapa sesi koneksi dengan komputer lainnya dan program di dalam jaringan. Informasi ini biasanya diketahui secara otomatis oleh sistem operasi *router* dan tidak membutuhkan konfigurasi.
2. Alamat IP (*Internet Protocol Address*) yang digunakan sebagai alamat identifikasi untuk tiap komputer dalam jaringan data, dari masing-masing *port*.

Setelah *router* mengetahui kedua informasi tersebut di atas, *router* akan menggabungkan informasi tersebut untuk membentuk entri-entri sebuah tabel. Tabel ini berada di dalam memori *router* dan tabel ini yang disebut sebagai tabel

routing. Tabel *routing* ini setidaknya memiliki dua *field* : alamat jaringan dan *hop* atau *link* berikutnya yang dapat berupa *ID* sebuah *interface*, misal e0 dan e1, atau alamat IP sebuah simpul tetangga. Setiap entri di dalam tabel *routing* disebut sebagai jalur. Pada Gambar 2.1.1 diperlihatkan sebuah *router* A yang memiliki dua *interface* Ethernet, dan satu *interface* ISDN (*Integrated Services Digital Network*), di mana *interface ethernet*0 (e0) diberi alamat IP 10.1.1.1/24 dan *interface ethernet*1 (e1) diberi alamat IP 10.1.2.1/24. Dalam praktiknya, tabel *routing* memuat jauh lebih banyak *field* atau informasi dari yang diperlihatkan pada Tabel 2.1.1 untuk *router* A pada Gambar 2.1.1.



Gambar 2.1.1 *Interface Router* A

Tabel 2.1.1 Tabel *Routing* pada *Router* A

Alamat Jaringan	*Hop* Berikutnya
10.1.1.0	e0
10.1.2.0	e1

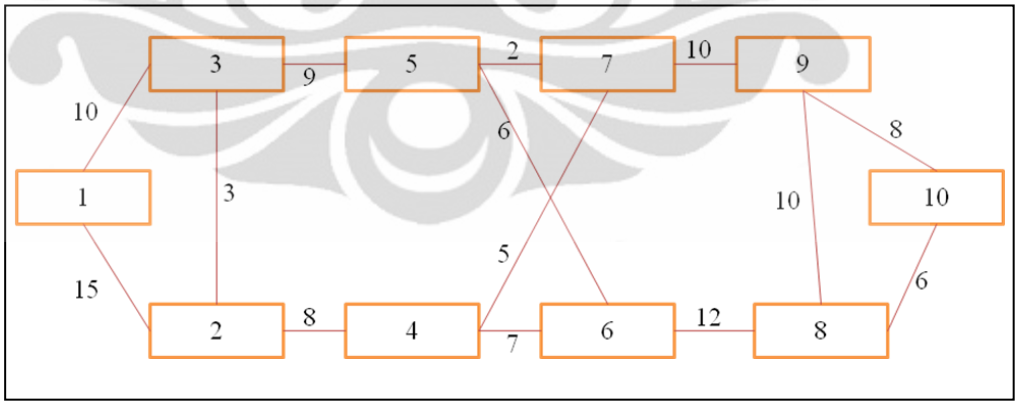
Tabel *routing* dapat juga berisi informasi lainnya, seperti informasi jalur yang diinginkan. *Router* membandingkan matriks yang berisi untuk menentukan rute optimal, dan matriks-matriks ini berbeda tergantung dengan desain algoritma *routing* yang digunakan. *Router* saling berkomunikasi dan memelihara tabel *routing*nya melalui transmisi pesan yang beragam. Dengan menganalisa

pembaharuan (*update*) *routing* dari semua *router* yang ada, sebuah *router* dapat membangun detail topologi jaringan.

Sementara mekanisme untuk menemukan jalur yang harus dilalui oleh suatu paket dari suatu *node* sumber ke *node* tujuan dilakukan oleh algoritma *routing*. Algoritma *routing* dibagi menjadi dua kelas utama, yaitu *non-adaptive* dan *adaptive* (D. A. Lubis, 2009). Algoritma *non-adaptive*, keputusan *routing* tidak berdasarkan penaksiran arus lalu lintas dan topologi sehingga jalur yang diambil oleh paket hanya bergantung pada basis sumber dan tujuan, tanpa memperdulikan status jaringan. Sedangkan algoritma *adaptive* dapat lebih mudah menyesuaikan pada situasi yang tidak konsisten, simpul yang masuk dalam jaringan atau kegagalan *link* atau perubahan lokal topologi.

2.2 Teori Graf

Suatu graf *G* adalah suatu pasangan terurut (*V*, *E*), dimana *V* adalah suatu himpunan tak kosong dan terbatas yang anggotanya disebut simpul dan *E* adalah himpunan busur (Foulds, 1992). Gambar 2.2.1 berikut menunjukkan contoh sebuah graf dengan 10 simpul, 14 busur, dan bobot masing-masing busur.



Gambar 2.2.1 Graf berbobot dengan 10 simpul

Busur dapat menunjukkan hubungan (relasi) sembarang seperti rute penerbangan, jalan raya, sambungan telepon, ikatan kimia, dan lain-lain. Sedangkan bobot pada busur menunjukkan biaya terkait hubungan tersebut. Graf

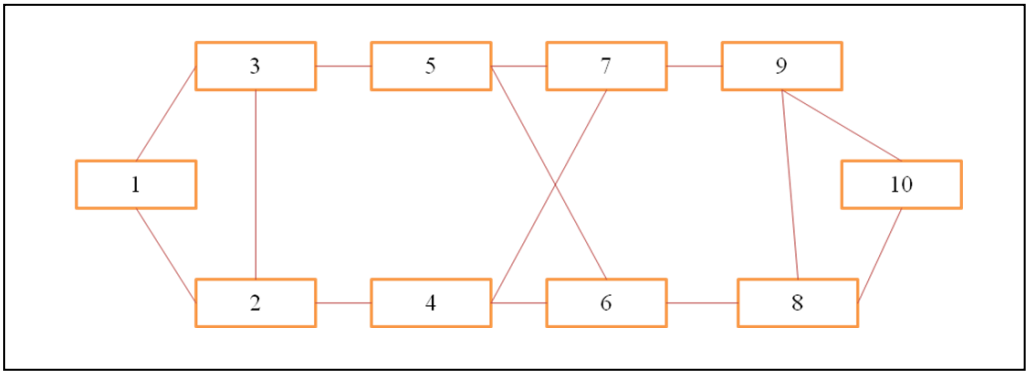
dinotasikan dengan $G(V, E)$. Pada umumnya graf digunakan untuk memodelkan suatu masalah sehingga menjadi lebih mudah, yaitu dengan cara merepresentasikan objek-objek tersebut.

2.2.1 Representasi Graf

Suatu graf dapat direpresentasikan ke beberapa bentuk. Matriks dapat digunakan untuk suatu graf. Hal ini tentu akan memudahkan program komputer yang berhubungan dengan graf.

2.2.1.1 Representasi Graf Tidak Berarah dan Tidak Berbobot dalam Matriks

Adjacency Matrix digunakan untuk menyatakan suatu graf dengan cara menyatakannya dalam jumlah busur yang menghubungkan setiap dua verteks. Jumlah baris dan kolom dari *adjacency matrix* sama dengan jumlah verteks atau simpul dalam graf. Pada graf tidak berarah dan tidak berbobot, jika antara dua buah verteks terhubung maka elemen matriks bernilai 1 (satu) dan sebaliknya bernilai 0 (nol) jika tidak terhubung. Dalam graf tidak berarah, jumlah busur yang menghubungkan verteks v_i dengan v_j selalu sama dengan jumlah busur yang menghubungkan v_j dengan v_i, sehingga jelas bahwa *adjacency matrix* untuk graf tidak berarah selalu merupakan matriks yang simetris ($a_{ij} = a_{ji} \quad \forall\, i,j$). *Adjacency matrix* untuk graf tidak berarah dan tidak berbobot ini juga dapat digunakan sebagai informasi keterhubungan antara satu *node* dengan *node* yang lainnya dalam suatu jaringan.



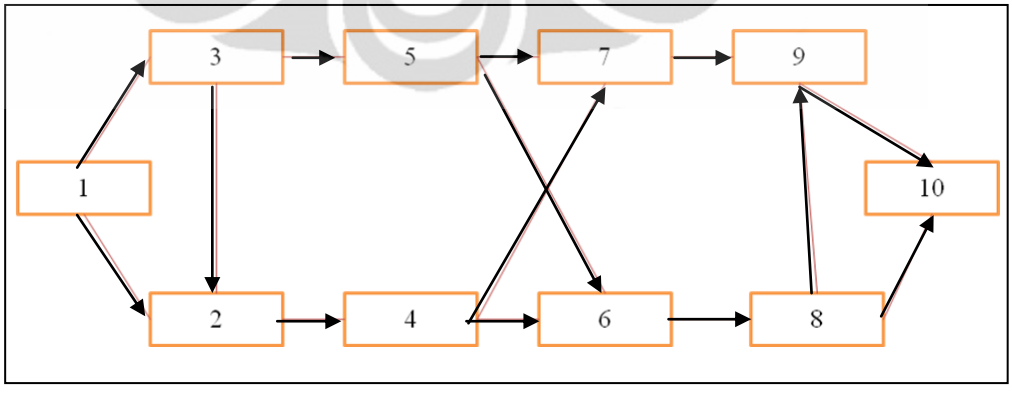
Gambar 2.2.2 Graf tidak berarah dengan 10 simpul

Adjacency Matrix M dari graf tidak berarah dan tidak berbobot pada Gambar 2.2.2 adalah sebagai berikut.

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

2.2.1.2 Representasi Graf Berarah dan Tidak Berbobot dalam Matriks

Adjacency Matrix pada graf berarah sebenarnya tidak terlalu berbeda dengan graf yang tidak berarah. Perbedaannya adalah *adjacency matrix* untuk graf berarah ini tidak selalu simetris.



Gambar 2.2.3 Graf berarah dengan 10 simpul

Adjacency Matrix M dari graf berarah dan tidak berbobot pada Gambar 2.2.3 di atas adalah sebagai berikut.

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

2.3 Algoritma Genetika

Algoritma genetika sebagai cabang dari algoritma evolusi merupakan metode *adaptive* yang biasa digunakan untuk memecahkan suatu pencarian nilai dalam sebuah masalah optimasi (E. Satriyanto, 2009). Algoritma ini didasarkan pada proses genetik yang ada dalam makhluk hidup, yaitu perkembangan generasi dalam sebuah populasi yang alami, secara lambat laun mengikuti prinsip seleksi alam. Dengan meniru teori evolusi ini, algoritma genetika dapat digunakan untuk mencari solusi permasalahan-permasalahan dalam dunia nyata.

Algoritma genetika pertama kali diperkenalkan sekitar tahun 1975 oleh John Holland dalam bukunya yang berjudul "*Adaption in Natural and Artificial Systems*" dan kemudian dikembangkan bersama murid dan rekan kerjanya (M. Obitko, 1998). Algoritma ini bekerja dengan sebuah populasi yang terdiri dari individu-individu yang masing-masing individu tersebut merepresentasikan sebuah solusi yang mungkin bagi masalah yang ada. Suatu individu direpresentasikan sebagai kumpulan gen yang disebut kromosom. Dengan menggunakan algoritma genetika, solusi yang dihasilkan belum tentu merupakan

solusi eksak dari masalah optimisasi yang diselesaikan. Beberapa definisi penting dalam algoritma genetika adalah sebagai berikut (E. Satriyanto, 2009):

- **Gen**

  Sebuah nilai yang menyatakan satuan dasar yang membentuk arti tertentu. Dalam algoritma genetika, gen ini dapat berupa nilai biner, *float*, integer maupun karakter, atau kombinatorial.

- **Kromosom**

  Merupakan gabungan gen-gen yang membentuk nilai tertentu.

- **Individu**

  Menyatakan satu nilai atau keadaan yang menyatakan salah satu solusi yang mungkin dari permasalahan yang diangkat. Dalam beberapa masalah yang dapat dipecahkan dengan algoritma genetika, individu ini dapat juga merupakan kromosom itu sendiri.

- **Populasi**

  Merupakan sekumpulan individu yang akan diproses bersama dalam satu siklus proses evolusi.

- **Generasi**

  Menyatakan satu siklus proses evolusi atau satu iterasi di dalam algoritma genetika.

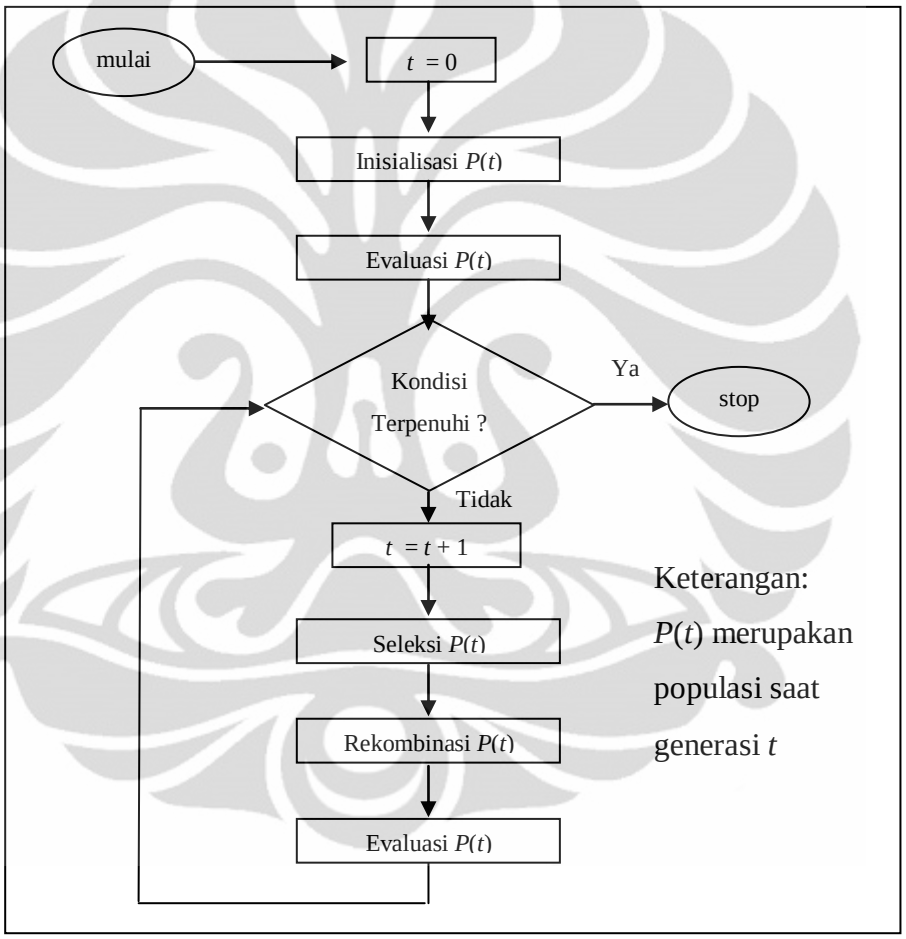
Dalam suatu masalah optimisasi, dikenal fungsi objektif atau fungsi tujuan yang merupakan fungsi pengevaluasi atau fungsi yang ingin dioptimalkan. Sedangkan dalam algoritma genetika, fungsi tersebut dinamakan fungsi *fitness.* Masing-masing individu memiliki nilai *fitness* tertentu.

Sebelum algoritma genetika dapat dijalankan, maka sebuah kode yang representatif untuk persoalan harus dirancang. Untuk ini maka suatu solusi dalam ruang permasalahan dikodekan dalam bentuk kromosom yang terdiri dari komponen genetik terkecil yaitu gen. Dengan teori evolusi dan teori genetika, di dalam penerapan algoritma genetika akan melibatkan beberapa operator, yaitu seleksi, *crossover*, dan mutasi.

Tahapan dari algoritma genetika yaitu pertama membentuk suatu populasi awal, kemudian populasi tersebut dievaluasi dengan fungsi *fitness* yang telah ditentukan, selanjutnya populasi tersebut diproses (direkombinasi) dengan

menggunakan operator-operator genetik seperti, seleksi, *crossover*, dan mutasi sehingga menghasilkan populasi baru untuk generasi berikutnya. Proses atau tahapan-tahapan tersebut diulang hingga mencapai kriteria berhenti tertentu. Kriteria berhenti dapat berupa batas generasi atau nilai optimal tertentu yang diinginkan.

Bentuk diagram alir standar algoritma genetika menurut Goldberg (D. E. Goldberg, 1989) dapat digambarkan seperti berikut :



Gambar 2.3.1 Bentuk diagram alir standar algoritma genetika

Bentuk standar algoritma genetika pada Gambar 2.3.1 dapat pula ditulis dalam bentuk *pseudocode* sebagai berikut :

Procedure Genetic Algorithm;

begin

$t := 0$;

Inisialisasi $P(t)$;

Evaluasi $P(t)$;

do while (Kondisi tak terpenuhi)

$t := t + 1$;

Select $P(t)$ from $P(t - 1)$;

Rekombinasi $P(t)$;

enddo;

end;

Dari prosedur di atas terlihat bahwa algoritma genetika memiliki waktu proses universal $O(n)$, dimana n merupakan banyaknya iterasi sampai kondisi berhenti terpenuhi.

Berikut akan dijelaskan mengenai masing-masing tahapan dari algoritma genetika.

2.3.1 Pembentukan Populasi Awal

Langkah awal dari algoritma genetika adalah menentukan representasi gen dan kromosom dari populasi. Terdapat beberapa jenis representasi, yaitu *string bit* (10011 ...), *array* bilangan *real* (65.55, -67.99, 77.33 ...), elemen permutasi (E1, E2, E5 ...), daftar aturan (R1, R2, R3 ...), dan representasi lainnya (E. Satriyanto, 2009). Terdapat beberapa teknik pengkodean dalam algoritma genetika, diantaranya (S. Lukas, T. Anwar, dan W. Yuliani, 2005) :

a. Pengkodean Biner (*binary encoding*)

Contohnya:

Tabel 2.3 Pengkodean Biner

Kromosom 1	**1 0 0 0 1 1 0 0 1 1 1 1 0 1 0 0 1 0 1 0**
Kromosom 2	**0 1 1 0 0 1 1 1 1 0 0 1 1 0 1 0 0 1 0 0**

b. Pengkodean Permutasi (*permutation encoding*)

Dalam pengkodean jenis ini tiap gen dalam kromosom merepresentasikan suatu urutan.

Contohnya:

Tabel 2.4 Pengkodean Permutasi

Kromosom 1	**1 2 3 4 5 6 7 8 9**
Kromosom 2	**2 8 9 3 7 4 6 1 5**

c. Pengkodean Nilai (*value encoding*)

Dalam pengkodean jenis ini tiap gen dalam kromosom adalah string dari suatu nilai.

Contohnya:

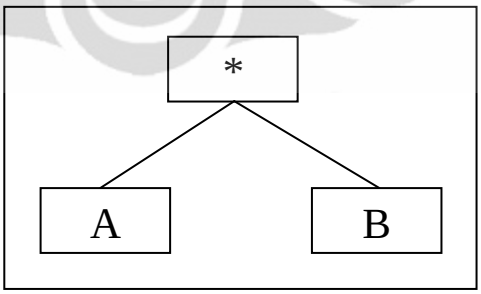
Tabel 2.5 Pengkodean Nilai

Kromosom 1	**1.232 5.324 0.455 2.388**
Kromosom 2	**ABCDEJAKAJAHAHAAHR**
Kromosom 3	**(left), (right), (back), (forward)**

d. Pengkodean Pohon (*tree encoding*)

Dalam pengkodean jenis ini biasanya digunakan untuk menyusun program atau ekspresi dari *genetic programming* (pemrograman genetik).

Contohnya: (*AB)



Gambar 2.3 Pengkodean Pohon

Populasi awal dibentuk dengan cara membangkitkan kromosom (individu) secara acak sebanyak ukuran populasi. Ukuran populasi akan berpengaruh pada

daerah pencarian solusi dan waktu komputasi yang dibutuhkan. Ukuran populasi sebaiknya disesuaikan dengan ukuran masalah yang akan diselesaikan.

2.3.2 Fungsi *Fitness*

Setelah populasi awal telah terbentuk, maka selanjutnya adalah mengevaluasi populasi tersebut dengan suatu fungsi *fitness*. Nilai *fitness* menyatakan seberapa baik nilai dari suatu kromosom (individu) atau solusi yang didapat. Nilai ini akan menjadi acuan dalam mencapai nilai optimal dalam algoritma genetika. Di dalam evolusi alam, individu yang bernilai *fitness* tinggi yang akan bertahan hidup, sementara individu bernilai *fitness* rendah tidak akan bertahan. Penentuan fungsi *fitness* biasanya disesuaikan dengan kondisi masalahnya.

2.3.3 Seleksi

Setelah membentuk populasi awal dan menentukan fungsi *fitness*, maka selanjutnya adalah melakukan seleksi pada populasi tersebut. Seleksi merupakan proses untuk menentukan individu mana saja yang akan dipilih untuk dilakukan rekombinasi dan bagaimana keturunan terbentuk dari individu-individu terpilih tersebut. Alat yang biasa digunakan adalah fungsi *fitness*.

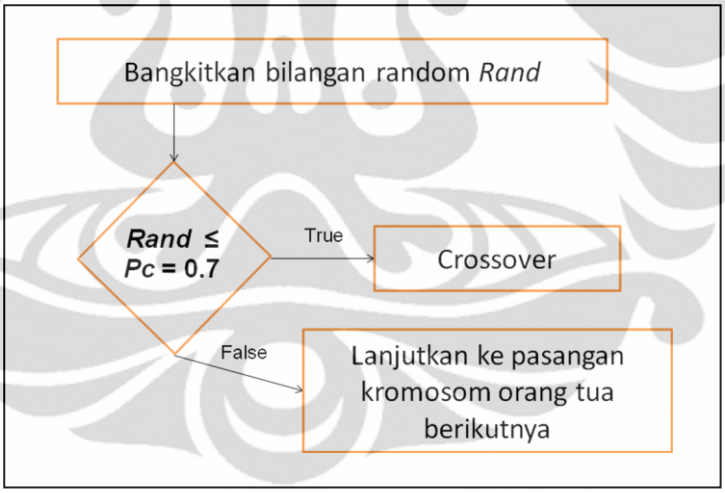
Individu yang terpilih atau terseleksi adalah individu dengan nilai *fitness* terbaik. Hal itu karena diharapkan dari individu yang baik akan didapatkan keturunan yang baik bahkan lebih baik dari individu tersebut sesuai dengan prinsip evolusi biologi. Ada beberapa macam metode seleksi yang ada pada algoritma genetika, diantaranya (S. Kusumadewi, 2003) :

a. Seleksi dengan Roda *Roulette*
b. Seleksi Lokal
c. Seleksi dengan Pemotongan
d. Seleksi dengan Turnamen

Dalam seleksi tersebut, proses seleksi akan menghasilkan populasi orang tua. Kemudian dari populasi orang tua akan dipilih sepasang kromosom orang tua yang selanjutnya mengalami proses *crossover*.

2.3.4 *Crossover*

Crossover adalah operator dari algoritma genetika yang melibatkan dua kromosom orang tua untuk membentuk kromosom baru (reproduksi). Tidak semua kromosom pada suatu populasi akan mengalami proses *crossover*. Kemungkinan suatu kromosom mengalami proses *crossover* adalah didasarkan pada probabilitas *crossover* (*Pc*) yang telah ditentukan. Probabilitas *crossover* menyatakan peluang suatu kromosom mengalami *crossover*. Proses *crossover* ini yang nantinya akan menghasilkan populasi kromosom anak (*offspring*). Pada Gambar 2.4 diilustrasikan diagram alir pada proses *crossover*.



Gambar 2.4 Diagram alir proses *crossover*

Salah satu teknik *crossover* yang dapat digunakan adalah *order crossover*. *Order crossover* (OX) diperkenalkan oleh Davis (W. S. E. Tanjung, 2010). Teknik ini diawali dengan membangkitkan dua bilangan acak. Kemudian gen yang berada diantara kedua bilangan acak tersebut (*substring*) disalin ke *offspring* dengan posisi yang sama pada masing-masing kromosom orang tua. Langkah berikutnya untuk mendapatkan *offspring* pertama adalah dengan mengurutkan gen

yang berada dalam kromosom orang tua kedua dengan urutan gen yang pertama adalah dari gen yang berada pada posisi setelah bilangan acak kedua yang telah dibangkitkan sebelumnya lalu diikuti oleh gen-gen yang berada pada posisi sebelum bilangan acak pertama dan diakhiri dengan gen-gen yang berada pada posisi diantara kedua bilangan acak. Kemudian gen yang telah diurutkan tadi dibandingkan dengan *offspring* pertama. Apabila gen yang terurut tersebut mengandung gen yang berada pada *offspring* pertama, maka abaikan gen tersebut dari urutan. Kemudian masukkan urutan yang baru saja didapat ke *offspring* pertama dengan cara memasukkan urutan gen pada posisi setelah bilangan acak kedua terlebih dahulu pada *offspring* pertama dan sisanya dimasukkan pada posisi sebelum bilangan acak pertama. Untuk menghasilkan *offspring* kedua dilakukan cara yang sama untuk kromosom orang tua pertama. Berikut adalah contoh dari *order crossover* (S. Lukas, T. Anwar, dan W. Yuliani, 2005):

Contoh 2.3

Misal terdapat 2 buah kromosom orang tua yang akan di-*crossover*, yaitu :

p1 = (1 2 3 4 5 6 7)

p2 = (1 5 4 6 2 3 7)

Kemudian bangun suatu bilangan acak yang akan menentukan posisi awal dan posisi akhir dari kromosom orang tua pertama (*p1*) yang akan ditukar dengan kromosom orang tua kedua (*p2*).

Misal yang terpilih adalah mulai dari gen nomor 4 sampai gen nomor 6 pada kromosom orang tua. Maka *substring* atau daerah yang akan di *crossover* adalah

p1 = (1 2 3 | **4 5** | 6 7)

p2 = (1 5 4 | **6 2** | 3 7)

Kemudian salin *substring* yang terpilih tersebut ke kromosom anak atau *offspring* menjadi

o1 = (x x x | **4 5** | x x)

o2 = (x x x | **6 2** | x x)

Kemudian posisi yang masih kosong diisi oleh gen-gen pada kromosom orang tua kedua (*p2*) yang telah diurutkan dimulai dari posisi atau batas akhir,

namun bila terdapat bilangan atau gen yang sama dengan gen pada daerah *substring* pada *offspring* pertama, maka gen pada *p2* tersebut dapat dilewatkan atau diabaikan, terakhir isi gen-gen yang masih kosong pada *offspring* pertama (*o1*) dimulai dari setelah batas akhir kemudian sebelum batas awal dengan sisa gen dari *p2* tadi sesuai urutan.

p2 = (3 7 1 5 4 6 2) => (3 7 1 6 2)

o1 = (1 6 2 | **4 5** | 3 7)

Dengan cara yang sama untuk menentukan *offspring* kedua.

Hasil akhirnya adalah :

o1 = (1 6 2 4 5 3 7)

o2 = (3 4 5 6 2 7 1)

Terlihat bahwa pada *offspring* hasil *crossover* di atas menjamin bahwa tidak ada gen yang berulang, sehingga nantinya akan didapat dalam hal ini jalur yang tidak mengandung *cycle*. Namun, pada *single point crossover* akan ada kemungkinan *offspring* yang terbentuk mengandung *cycle* (N. K. Cauvery dan K. V. Viswanatha, 2009). Contoh 2.3.1 akan memberikan contoh *single point crossover* yang *offspring*-nya mengandung *cycle*.

Contoh 2.3.2

Misal terpilih 2 kromosom orang tua :

p1 = (1 2 3 4 5 6)

p2 = (1 5 4 2 3 6)

Misal terpilih posisi gen yang akan di-*crossover* adalah posisi ke-5, maka *offspring* nya adalah

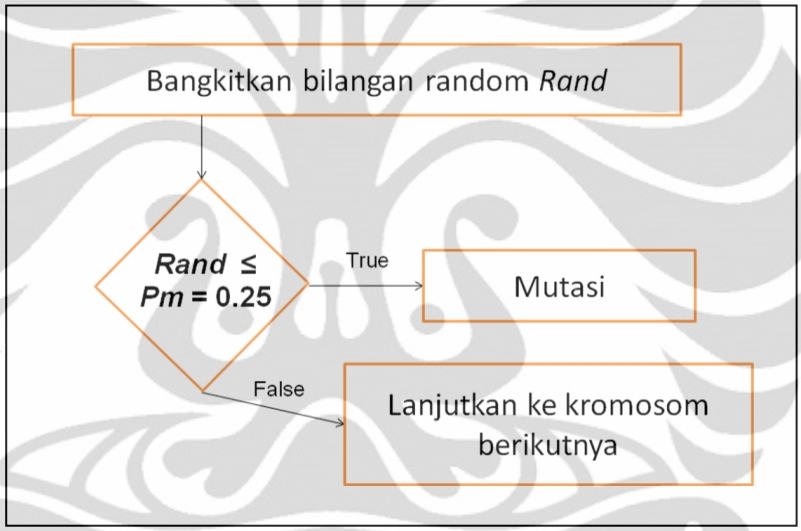
o1 = (1 2 3 4 **3** 6)

o2 = (1 5 4 2 **5** 6)

Terlihat jelas bahwa *offspring* yang terbentuk terdapat gen yang berulang, sehingga nantinya jalur akan mengandung *cycle*.

2.3.5 Mutasi

Tahapan selanjutnya setelah *crossover* adalah mutasi. Operator mutasi ini berperan untuk mengembalikan solusi optimal yang dapat hilang akibat proses *crossover* sebelumnya. Operator mutasi dapat mencegah terjadinya solusi optimum lokal (N. Murniati, 2009). Banyaknya kromosom yang akan mengalami mutasi dihitung berdasarkan probabilitas mutasi (*Pm*) yang telah ditentukan terlebih dahulu. Jika probabilitas mutasi yang digunakan adalah 0% maka tidak akan ada kromosom yang mengalami mutasi pada populasi tersebut. Pada Gambar 2.5 diilustrasikan diagram alir pada proses mutasi.



Gambar 2.5 Diagram alir proses mutasi

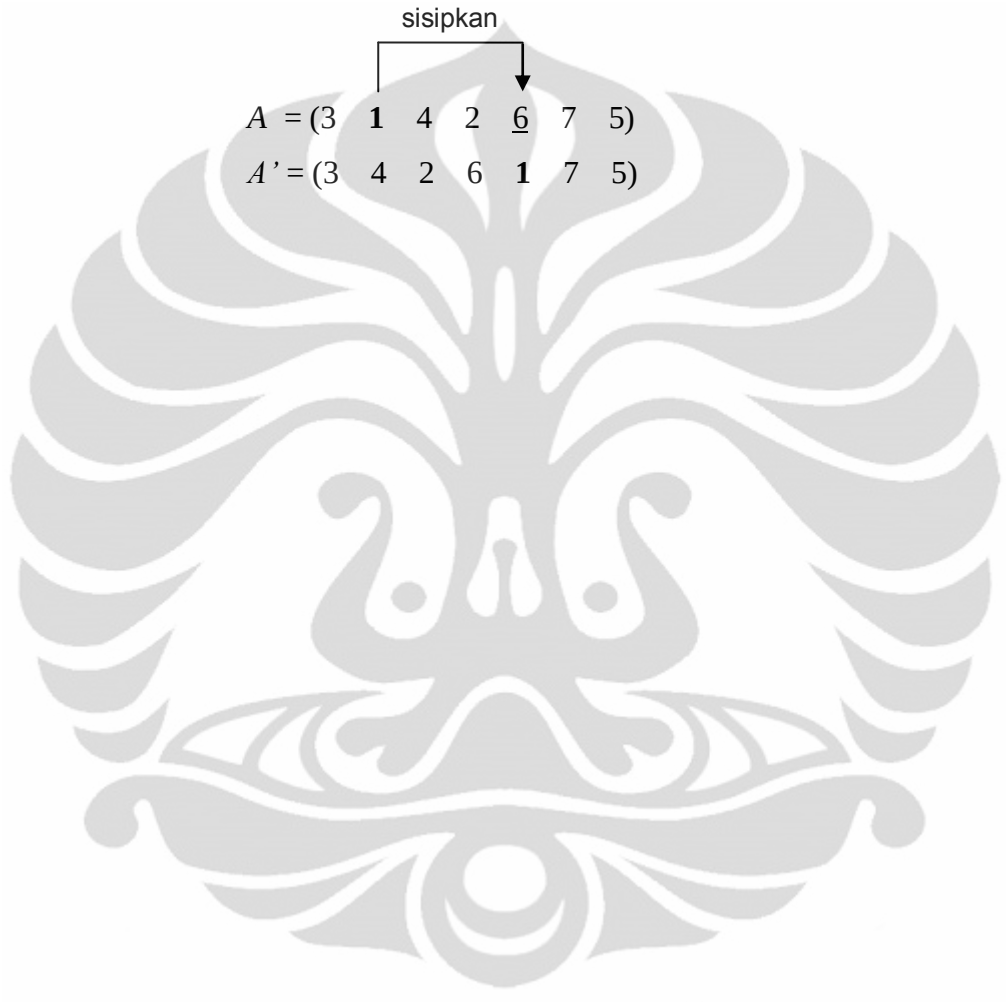
Salah satu teknik mutasi yang dapat digunakan adalah teknik *insertion mutation*. Teknik ini diawali dengan memilih dua bilangan acak, kemudian gen yang berada pada posisi bilangan acak pertama di sisipkan ke posisi bilangan acak kedua (S. Lukas, T. Anwar, dan W. Yuliani, 2005). Berikut adalah contoh dari *insertion mutation*:

Contoh 2.4

Misal kromosom A = (3 1 4 2 6 7 5) terpilih untuk dimutasi.

Misal bilangan acak yang pertama terpilih adalah 2 dan bilangan acak yang kedua adalah 5.

Selanjutnya sisipkan gen pada posisi ke-2 dalam hal ini 1 ke posisi 5.

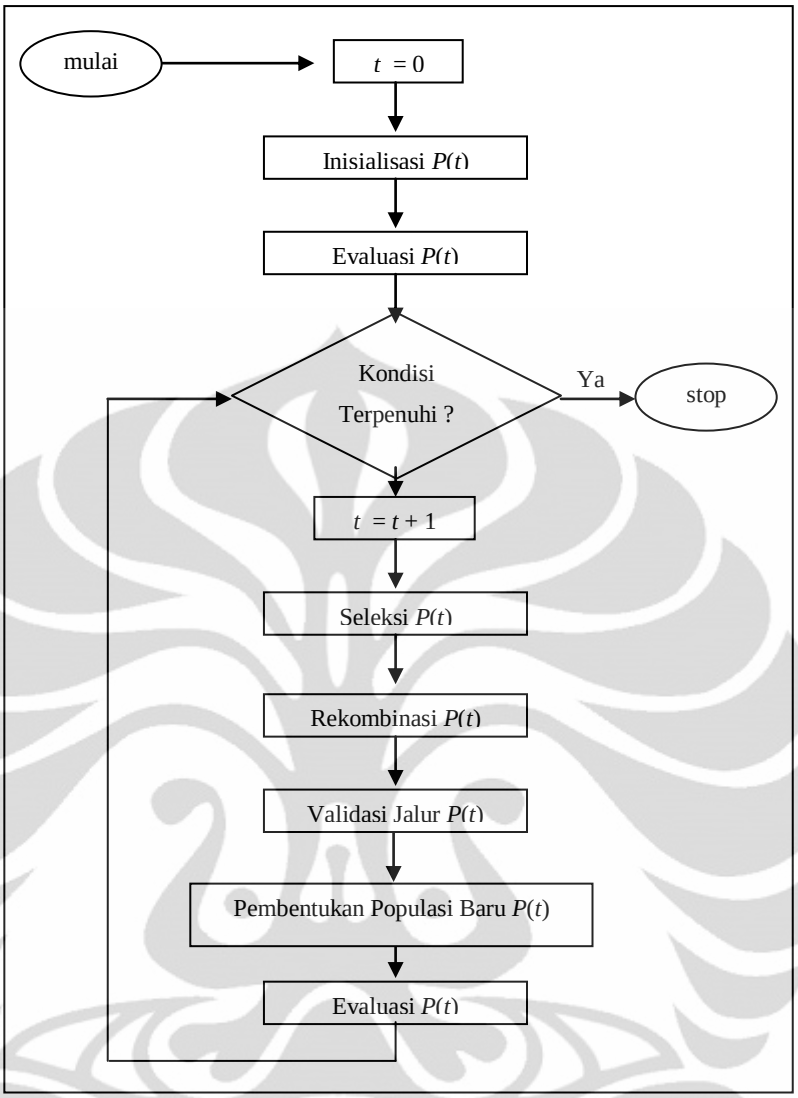


BAB 3
ALGORITMA GENETIKA DALAM MENYELESAIKAN MASALAH JALUR TERPENDEK

Pada bab ini akan dibahas mengenai penggunaan algoritma genetika untuk menyelesaikan masalah jalur terpendek. Beberapa proses penting yang harus dilakukan untuk mengimplementasikan algoritma genetika dalam mencari jalur terpendek yaitu sebagai berikut (F. Saptono, I. Mutakhiroh, T. Hidayat, dan A. Fauziyah, 2007) :

a. Pengkodean kromosom
b. Inisialisasi populasi
c. Menentukan nilai *fitness*
d. Proses seleksi
e. Proses *crossover*
f. Proses mutasi

Representasi kromosom pada algoritma genetika untuk masalah jalur terpendek akan dibahas pada Subbab 3.1, sementara pembentukan atau inisialisasi populasi awal dibahas pada Subbab 3.2. Evaluasi fungsi *fitness* dari tiap kromosom akan dibahas pada Subbab 3.3. Kemudian proses evolusi yang melibatkan operator evolusi seleksi, *crossover*, dan mutasi masing-masing akan dibahas pada Subbab 3.4, 3.5, 3.6. Sementara pembentukan populasi untuk generasi berikutnya akan dibahas pada Subbab 3.7 dan pada Subbab 3.8 akan dibahas mengenai kriteria berhenti yang digunakan. Algoritma genetika untuk masalah jalur terpendek beserta diagram alirnya akan diberikan pada Subbab 3.9. Berikut adalah diagram alir dari keseluruhan proses :



Gambar 3.1 Diagram alir keseluruhan proses

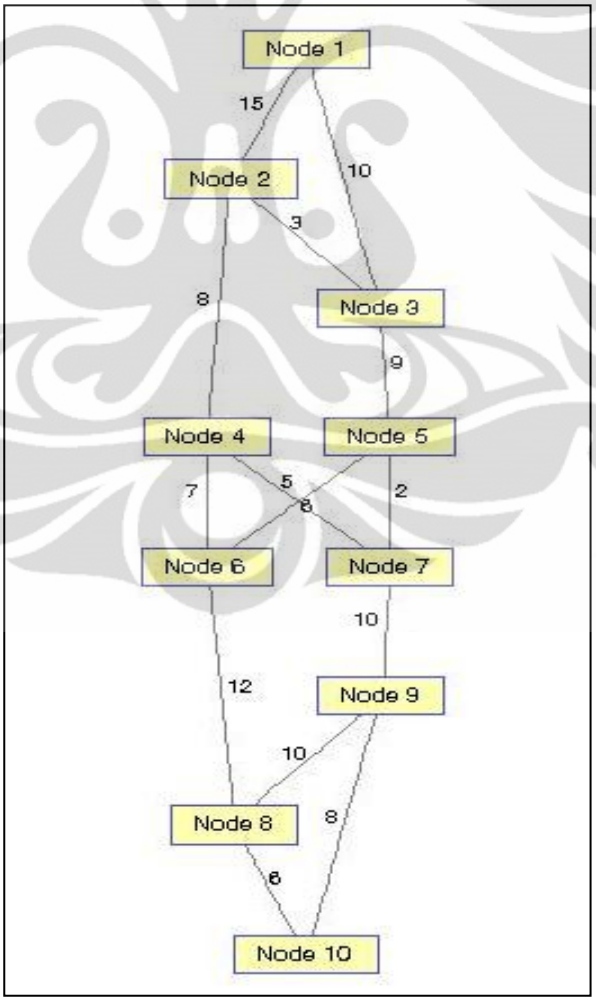
3.1 Pengkodean Atau Representasi Kromosom

Pada algoritma genetika yang dijelaskan pada skripsi ini, jenis representasi kromosom yang digunakan adalah representasi permutasi atau pengkodean permutasi. Dengan mengkodekan simpul-simpul pada jaringan dengan bilangan bulat positif 1, 2, 3, …, *n*, di mana *n* adalah banyaknya simpul pada jaringan. Tiap kode dari simpul dapat dianggap sebagai gen pada kromosom, sehingga kromosom merupakan untaian kode-kode dari simpul pada jaringan yang tidak berulang dan merepresentasikan suatu urutan atau jalur.

Pada masalah jalur terpendek pada jaringan data, panjang kromosom atau banyaknya gen sama dengan banyaknya simpul. Untuk masalah jalur terpendek, kromosom dirancang dengan menggunakan prinsip pengacakan, dimana untuk perhitungan atau penentuan nilai *fitness* dilakukan hanya untuk jalur yang dilalui dari *node* sumber ke *node* tujuan, dengan sisa kromosom (simpul yang tidak dilalui) berfungsi sebagai pelengkap (F. Saptono dan T. Hidayat, 2007).

Contoh 3.1 :

Berikut ini contoh persoalan masalah jalur terpendek dari suatu jaringan data yang terdapat 10 simpul. *Node* sumber adalah *node* 1 dan *node* akhir adalah *node* 10. Karena banyaknya simpul 10, maka panjang kromosom atau banyaknya gen dalam satu kromosom adalah 10.



Gambar 3.2 Topologi jaringan untuk simulasi (G. Nagib dan W. G. Ali, 2010)

Tabel 3.1 memperlihatkan jalur yang terbentuk dengan masing-masing representasi kromosomnya. Gen-gen yang ditebalkan merupakan representasi jalur yang valid, sementara gen-gen lain yang tidak ditebalkan hanya sebagai pelengkap.

Tabel 3.1 Jalur yang berkorespondensi dengan masing-masing kromosom

Jalur	Representasi Kromosom
1-3-5-7-9-10	**1-3-5-7-9-10**-2-4-6-8
1-3-5-7-9-8-10	**1-3-5-7-9-8-10**-2-4-6
1-2-4-6-8-10	**1-2-4-6-8-10**-3-5-7-9

3.2 Inisialisasi Populasi

Tahap ini bertujuan untuk membangkitkan sebuah populasi yang berisi sejumlah kromosom yang telah ditentukan banyaknya.

Misal banyaknya kromosom dalam populasi awal telah ditentukan sebanyak 10 kromosom, misal kromosom yang terbentuk terlihat seperti pada Tabel 3.2. Kromosom dibentuk secara acak dengan menetapkan gen pertama sebagai *node* sumber, dalam hal ini adalah *node* 1.

Pada tahap atau proses ini sebenarnya dapat ditambahkan algoritma baru untuk proses pencarian lokal agar algoritma genetika dapat menjamin solusi optimum global. Karena itu penulis memberikan saran untuk pengembangan skripsi ini pada akhir bab, yaitu perlu adanya tambahan algoritma untuk membangkitkan beberapa jalur yang mungkin (valid) dari *node* sumber ke *node* tujuan agar dapat muncul kromosom unggul pada populasi awal dan sebagai akibatnya dapat mengurangi waktu komputasi.

Tabel 3.2 Populasi awal yang terbentuk

Kromosom	Representasi Kromosom
kromosom 1	**1-2-4-7-9-8-10**-3-6-5
kromosom 2	**1-2-4-7-9-10**-5-8-6-3
kromosom 3	**1-3-5-6-8-9-10**-4-7-2
kromosom 4	**1-3-5-7-9-10**-4-6-2-8
kromosom 5	**1-3-5-7-9-10**-6-4-8-2
kromosom 6	**1-2-4-7-9-10**-5-3-8-6
kromosom 7	**1-3-2-4-7-5-6-8-9-10**
kromosom 8	**1-2-4-6-8-10**-9-7-5-3
kromosom 9	**1-2-4-7-9-10**-3-8-5-6
kromosom 10	**1-2-4-7-5-6-8-9-10**-3

3.3 Evaluasi Fungsi *Fitness*

Fungsi *fitness* digunakan untuk menentukan seberapa baik individu yang direpresentasikan oleh suatu kromosom. Dalam kasus ini permasalahan jalur terpendek yaitu untuk mencari jarak terpendek dari 10 *node* dan 14 busur yang telah disebutkan pada Contoh 3.1. Nilai *fitness* yang dapat digunakan adalah 1 / total jarak (satu per total jarak). Dalam hal ini yang dimaksud total jarak adalah jumlah jarak antara satu *node* dengan *node* lainnya yang dapat dilalui. Semakin tinggi nilai *fitness* dari suatu individu atau kromosom, maka semakin baik individu tersebut (D. E. Goldberg, 1989). Nilai *fitness* ini juga bergantung pada keabsahan dari jalur yang terkandung dalam kromosom yang bersangkutan. Jika ada kromosom yang memiliki jalur dari *node* sumber ke *node* tujuan yang tidak valid, maka nilai *fitness* akan sama dengan nol, namun sebaliknya jika kromosom memiliki jalur dari *node* sumber ke *node* tujuan yang valid, maka nilai *fitness* akan sama dengan nilai dari fungsi *fitness* yang telah ditentukan. Berikut adalah penentuan nilai *fitness* pada skripsi ini (G. Nagib dan W. G. Ali, 2010):

$$F = \begin{cases} \dfrac{1}{\sum_{i=1}^{n-1} C_i\,(g_i, g_{i+1})} & ; \text{ Jalur valid} \\ 0 & ; \text{ Jalur tidak valid} \end{cases}$$

di mana $C_i\,(g_i, g_{i+1})$ adalah *cost* antara gen g_i dan gen tetangganya g_{i+1} dalam kromosom dari ***n*** gen (simpul).

Tabel 3.3 memperlihatkan kromosom dari Tabel 3.2 dengan masing-masing nilai *fitness* dan *cost* dari jalur yang terkandung di dalamnya.

Tabel 3.3 Nilai *fitness* dari masing-masing kromosom

Kromosom	Nilai *Cost* (total jarak)	Nilai *Fitness*
kromosom 1	15+8+5+10+10+6 = 54	**0.0185**
kromosom 2	15+8+5+10+8 = 46	**0.0217**
kromosom 3	10+9+6+12+10+8 = 55	**0.0182**
kromosom 4	10+9+2+10+8 = 39	**0.0256**
kromosom 5	10+9+2+10+8 = 39	**0.0256**
kromosom 6	15+8+5+10+8 = 46	**0.0217**
kromosom 7	10+3+8+5+2+6+12+10+8 = 64	**0.0156**
kromosom 8	15+8+7+12+6 = 48	**0.0208**
kromosom 9	15+8+5+10+8 = 46	**0.0217**
kromosom 10	15+8+5+2+6+12+10+8 = 66	**0.0152**

Pada Tabel 3.3 di atas terlihat bahwa kromosom 4 dan 5 memiliki nilai *fitness* yang tertinggi, sedangkan kromosom 10 memiliki nilai *fitness* yang terendah.

3.4 Seleksi

Setelah terbentuk populasi awal, selanjutnya hasil populasi tersebut akan diseleksi. Metode seleksi yang digunakan dalam algoritma genetika untuk pencarian jalur terpendek ini adalah seleksi roda *roulette*.

Dalam metode roda *roulette*, proses seleksi individu diibaratkan seperti dalam permainan judi roda *roulette*. Dimana pemain akan memutar roda yang telah terpartisi menjadi beberapa bagian untuk mendapatkan suatu hadiah. Kaitannya dengan metode seleksi yang dibahas ini adalah suatu kromosom diibaratkan sebagai hadiah (N. Murniati, 2009). Partisi-partisi pada roda *roulette* merupakan interval dari nilai kumulatif probabilitas masing-masing kromosom. Kemudian proses memutar roda dinyatakan dengan menentukan suatu bilangan acak pada interval [0, 100]. Pada proses seleksi ini, suatu kromosom kadang terpilih lebih dari sekali dan lebih dari satu. Kromosom-kromosom yang terpilih tersebut akan membentuk populasi orang tua.

Adapun tahapan dari proses seleksi sebagai berikut (E. Satriyanto, 2009):

Tahap 1: Hitung nilai *fitness* dari masing-masing kromosom.

Tahap 2: Hitung total *fitness* semua individu atau kromosom.

Tahap 3: Hitung probabilitas dan nilai kumulatif probabilitas masing-masing kromosom.

Tahap 4: Dari probabilitas tersebut, hitung jatah masing-masing individu pada angka 0 sampai 100, atau dengan kata lain menentukan interval kumulatif probabilitas masing-masing kromosom.

Tahap 5: Bangkitkan bilangan acak antara 0 sampai 100.

Tahap 6: Dari bilangan acak yang dihasilkan, tentukan kromosom mana yang terpilih dalam proses seleksi menurut interval yang bersesuaian yang telah ditentukan sebelumnya pada tahap 4.

Tabel 3.4 memperlihatkan hasil seleksi dengan bilangan acak yang dibangkitkan.

Tabel 3.4 Populasi kromosom orang tua

Bilangan acak	Kromosom Orangtua	Kromosom	Representasi Kromosom
85.2042	Orangtua 1	kromosom 9	1-2-4-7-9-10-3-8-5-6
82.512	Orangtua 2	kromosom 9	1-2-4-7-9-10-3-8-5-6
90.505	Orangtua 3	kromosom 9	1-2-4-7-9-10-3-8-5-6
80.2401	Orangtua 4	kromosom 8	1-2-4-6-8-10-9-7-5-3
60.1012	Orangtua 5	kromosom 6	1-2-4-7-9-10-5-3-8-6
95.201	Orangtua 6	kromosom 10	1-2-4-7-5-6-8-9-10-3
39.2014	Orangtua 7	kromosom 4	1-3-5-7-9-10-4-6-2-8
7.5001	Orangtua 8	kromosom 1	1-2-4-7-9-8-10-3-6-5
62.5101	Orangtua 9	kromosom 6	1-2-4-7-9-10-5-3-8-6
63.2431	Orangtua 10	kromosom 6	1-2-4-7-9-10-5-3-8-6

3.5 *Crossover*

Salah satu komponen yang paling penting dalam algoritma genetika adalah *crossover* atau pindah silang. *Crossover* merupakan suatu proses persilangan sepasang kromosom orang tua untuk menghasilkan *offspring* yang akan menjadi individu pada populasi di generasi berikutnya. *Offspring* yang dihasilkan dari proses *crossover* diharapkan mewarisi sifat-sifat unggul yang dimiliki oleh kromosom orang tua. Pindah silang pada masalah jalur terpendek ini menggunakan *Order Crossover*. Banyaknya kromosom yang di-*crossover* dipengaruhi oleh parameter probabilitas *crossover* (*Pc*).

Adapun tahapan dari proses *crossover* sebagai berikut (S. Lukas, T. Anwar, dan W. Yuliani, 2005) :

Tahap 1: Pilih dua kromosom berbeda pada populasi orang tua secara berurutan.

Tahap 2: Pilih *substring* dari orang tua secara berurut.

Tahap 3: Salin *substring* dan *node* sumber ke *offspring* yang akan digenerasi dengan posisi yang sama.

Tahap 4: Abaikan gen dengan nilai yang sama dengan yang sudah ada di tahap 2.

Tahap 5: Tempatkan sisa *substring* kromosom orang tua ke *offspring* setelah daerah *substring* dari *offspring* dengan urutan yang sama.

Setelah *offspring* terbentuk dari proses *crossover* maka selanjutnya adalah memvalidasi jalur yang terkandung di dalamnya, karena bisa jadi *offspring* yang terbentuk merepresentasikan jalur yang tidak valid (R. Kumar dan M. Kumar, 2010). Jika jalur tersebut tidak valid, maka *offspring* tersebut tidak akan menjadi bagian dari generasi berikutnya. Sebaliknya, jika jalur valid, maka *offspring* tersebut dapat menjadi bagian dari generasi berikutnya. Misalkan *offspring* yang terbentuk adalah sebagai berikut:

Tabel 3.5 Populasi *offspring* hasil *crossover*

Offspring	Representasi Kromosom	Representasi Jalur
Offspring 1	1-4-6-9-7-10-3-8-5-2	Tidak valid
Offspring 2	1-2-4-3-8-10-9-7-5-6	Tidak valid
Offspring 3	1-3-4-6-8-10-9-5-2-7	Tidak valid
Offspring 4	1-8-4-7-9-10-3-5-2-6	Tidak valid
Offspring 5	1-9-10-7-5-6-3-8-2-4	Tidak valid
Offspring 6	1-5-6-7-9-10-8-3-2-4	Tidak valid
Offspring 7	1-2-4-7-9-8-10-3-6-5	Valid
Offspring 8	1-3-5-7-9-10-4-6-2-8	Valid
Offspring 9	1-2-4-7-9-10-5-3-8-6	Valid
Offspring 10	1-2-4-7-9-10-5-3-8-6	Valid

Terlihat pada Tabel 3.5 di atas bahwa *offspring* ke-7 sampai 10 mengandung jalur yang valid, dalam hal ini adalah jalur yang dapat mencapai *node* tujuan yaitu *node* 10 dari *node* sumber yaitu *node* 1. Sementara *offspring* 1 sampai *offspring* 6 mengandung jalur yang tidak valid. *Offspring* yang mengandung jalur tidak valid tidak akan digunakan untuk generasi selanjutnya.

3.6 Mutasi

Mutasi adalah suatu proses yang dilakukan untuk mempertahankan keanekaragaman genetik populasi. Hal tersebut dilakukan untuk mencegah populasi terjebak dalam solusi optimal lokal.

Daftar populasi baru hasil *crossover* dipilih secara acak untuk dilibatkan dalam proses mutasi. Pada algoritma genetika, mutasi memainkan peran penting, yaitu menggantikan gen-gen yang hilang dari populasi selama proses seleksi atau mengembalikan kromosom optimal yang hilang akibat proses *crossover* (R. Kumar dan M. Kumar, 2010). Dan memunculkan kromosom yang tidak ditampilkan pada populasi awal yang bisa jadi lebih baik dari kromosom pada populasi saat itu.

Adapun tahapan dari proses mutasi sebagai berikut (S. Lukas, T. Anwar, dan W. Yuliani, 2005) :

Tahap 1: Pilih satu kromosom pada populasi anak hasil *crossover*.

Tahap 2: Pilih 2 posisi secara acak. Posisi pertama digunakan untuk menandakan gen mana yang akan dimutasi atau disisipkan ke posisi kedua.

Tahap 3: Sisipkan gen pada posisi pertama ke posisi kedua.

Setelah *offspring* hasil mutasi terbentuk maka selanjutnya adalah memvalidasi jalur yang terkandung di dalamnya dengan teknik yang sama pada setelah *crossover*, karena bisa jadi *offspring* hasil mutasi tersebut merepresentasikan jalur yang tidak valid (R. Kumar dan M. Kumar, 2010). Jika jalur tersebut tidak valid, maka *offspring* tersebut tidak akan menjadi bagian dari generasi berikutnya. Sebaliknya, jika jalur valid, maka *offspring* tersebut dapat menjadi bagian dari generasi berikutnya.

Tabel 3.6 Populasi *offspring* hasil mutasi

Offspring	Representasi Kromosom	Kromosom Hasil Mutasi	Representasi Jalur
Offspring 1	1-4-6-9-7-10-3-8-5-2	1-4-6-9-7-10-3-8-5-2	Tidak valid
Offspring 2	1-2-4-3-8-10-9-7-5-6	1-6-2-4-3-8-10-9-7-5	Tidak valid
Offspring 3	1-3-4-6-8-10-9-5-2-7	1-3-4-6-8-10-9-5-2-7	Tidak valid
Offspring 4	1-8-4-7-9-10-3-5-2-6	1-4-7-9-10-8-3-5-2-6	Tidak valid
Offspring 5	1-9-10-7-5-6-3-8-2-4	1-9-10-7-5-6-3-8-2-4	Tidak valid
Offspring 6	1-5-6-7-9-10-8-3-2-4	1-5-6-7-9-10-8-3-2-4	Tidak valid
Offspring 7	1-2-4-7-9-8-10-3-6-5	1-2-4-7-9-8-10-3-6-5	Valid
Offspring 8	1-3-5-7-9-10-4-6-2-8	1-3-5-7-9-10-4-6-2-8	Valid
Offspring 9	1-2-4-7-9-10-5-3-8-6	1-2-4-7-9-10-5-3-8-6	Valid
Offspring 10	1-2-4-7-9-10-5-3-8-6	1-2-4-7-9-10-5-3-8-6	Valid

Terlihat pada Tabel 3.6 di atas bahwa *offspring* yang mengalami mutasi hanya pada *offspring* 2 dan 4. *Offspring* hasil mutasi yang memiliki jalur tidak valid tidak akan digunakan pada generasi selanjutnya.

3.7 Pembentukan Populasi Untuk Generasi Berikutnya

Langkah awal dari pemilihan individu untuk generasi berikutnya adalah dengan menggabungkan semua kromosom orang tua dan semua kromosom anak baik yang mengalami mutasi maupun yang tidak mengalami mutasi. Selanjutnya hitung nilai *fitness* gabungan kromosom tersebut. Lalu *sorting* kromosom dari yang memiliki *fitness* tertinggi sampai terendah. Terakhir ambil kromosom yang memiliki nilai *fitness* tertinggi sebanyak ukuran populasi yang telah ditentukan di awal.

Tabel 3.7 Gabungan populasi yang telah diurutkan berdasarkan nilai *fitness*

Kromosom	Representasi Kromosom	Nilai *Fitness*
Kromosom 4	1-3-5-7-9-10-4-6-2-8	**0.0256**
Kromosom 5	1-3-5-7-9-10-6-4-8-2	**0.0256**
Offspring 8	1-3-5-7-9-10-4-6-2-8	**0.0256**
Kromosom 2	1-2-4-7-9-10-5-8-6-3	**0.0217**
Kromosom 6	1-2-4-7-9-10-5-3-8-6	**0.0217**
Kromosom 9	1-2-4-7-9-10-3-8-5-6	**0.0217**
Offspring 9	1-2-4-7-9-10-5-3-8-6	**0.0217**
Offspring 10	1-2-4-7-9-10-5-3-8-6	**0.0217**
Kromosom 8	1-2-4-6-8-10-9-7-5-3	**0.0208**
Kromosom 1	1-2-4-7-9-8-10-3-6-5	**0.0185**
Offspring 7	1-2-4-7-9-8-10-3-6-5	**0.0185**
Kromosom 3	1-3-5-6-8-9-10-4-7-2	**0.0182**
Kromosom 7	1-3-2-4-7-5-6-8-9-10	**0.0156**
Kromosom 10	1-2-4-7-5-6-8-9-10-3	**0.0152**
Offspring 1	1-4-6-9-7-10-3-8-5-2	**0**
Offspring 2	1-6-2-4-3-8-10-9-7-5	**0**
Offspring 3	1-3-4-6-8-10-9-5-2-7	**0**
Offspring 4	1-4-7-9-10-8-3-5-2-6	**0**
Offspring 5	1-9-10-7-5-6-3-8-2-4	**0**
Offspring 6	1-5-6-7-9-10-8-3-2-4	**0**

Tabel 3.8 Pembentukan populasi untuk generasi berikutnya

Kromosom	Representasi Kromosom	Nilai *Fitness*
Kromosom 4	1-3-5-7-9-10-4-6-2-8	**0.0256**
Kromosom 5	1-3-5-7-9-10-6-4-8-2	**0.0256**
Offspring 8	1-3-5-7-9-10-4-6-2-8	**0.0256**
Kromosom 2	1-2-4-7-9-10-5-8-6-3	**0.0217**
Kromosom 6	1-2-4-7-9-10-5-3-8-6	**0.0217**
Kromosom 9	1-2-4-7-9-10-3-8-5-6	**0.0217**
Offspring 9	1-2-4-7-9-10-5-3-8-6	**0.0217**
Offspring 10	1-2-4-7-9-10-5-3-8-6	**0.0217**
Kromosom 8	1-2-4-6-8-10-9-7-5-3	**0.0208**
Kromosom 1	1-2-4-7-9-8-10-3-6-5	**0.0185**

Terlihat pada Tabel 3.7 di atas kromosom pada populasi awal digabung dengan populasi *offspring* terakhir hasil mutasi yang telah diurutkan berdasarkan nilai *fitness*. Pada Tabel 3.8 diperlihatkan kromosom-kromosom yang terambil dari gabungan populasi yang telah diurutkan sebelumnya sebanyak ukuran populasi yang telah ditentukan untuk populasi baru pada generasi berikutnya.

3.8 Kriteria Berhenti

Proses-proses pada algoritma genetika akan terus berulang sampai mencapai suatu kriteria berhenti tertentu. Kriteria berhenti yang digunakan pada algoritma genetika untuk pencarian jalur terpendek ini adalah *generations*, yaitu algoritma genetika akan berhenti setelah mencapai batas generasi yang telah ditentukan.

3.9 Algoritma Genetika Untuk Masalah Jalur Terpendek

Berdasarkan Subbab 3.1 sampai dengan Subbab 3.8, maka secara umum untuk menyelesaikan masalah jalur terpendek digunakan algoritma genetika sebagai berikut:

begin

Baca data (topologi jaringan);

Tentukan *node* sumber dan *node* tujuan;

Tentukan parameter algoritma genetika;

Bentuk populasi awal secara acak sesuai ukuran populasi *popsize*;

for generasi = 1 **to** max_generasi **do**

Hitung nilai *fitness* pada tiap kromosom pada populasi;

Bentuk populasi orang tua dengan seleksi;

Terapkan *crossover* dan mutasi (jika memenuhi kriteria);

Bentuk populasi baru untuk generasi selanjutnya;

endfor;

end;

BAB 4
IMPLEMENTASI DAN BEBERAPA HASIL PERCOBAAN

4.1 Implementasi

Pada Bab 3 telah dijelaskan penggunaan algoritma genetika untuk menyelesaikan masalah jalur terpendek. Pada bab ini akan dijelaskan mengenai implementasi algoritma genetika tersebut menggunakan MATLAB 7.5. *Listing* program dari metode tersebut dapat dilihat pada Lampiran 1.

Masukan dari program adalah :

- File *input* yang berisi data jaringan seperti *cost* tiap jalur dan topologi jaringan. Format *file input* dapat dilihat pada Lampiran 2.
- *Node* sumber (*source node*)
- *Node* tujuan (*destination node*)
- Parameter-parameter algoritma genetika, yaitu ukuran populasi (*popsize*), batas generasi maksimum (banyak iterasi), probabilitas *crossover*, serta probabilitas mutasi.

Berikut adalah fungsi-fungsi yang digunakan pada program beserta dengan penjelasannya :

- gaspp

  Algoritma genetika untuk masalah jalur terpendek secara keseluruhan akan dijalankan oleh fungsi gaspp. Fungsi ini akan memanggil beberapa fungsi lain seperti, totalCost, randChrom, crossover, mutation, elitism, dan validatePath. Di dalam fungsi ini pula terdapat operator genetik, seperti reproduksi, *crossover*, dan mutasi.

- fitnessV

  Fungsi untuk menentukan nilai *fitness* untuk masing-masing kromosom pada suatu populasi.

- totalCost

  Fungsi untuk menghitung total *cost* semua jalur yang *valid* atau dapat dilalui pada jaringan dari *node* sumber ke *node* tujuan.

- initChrom

  Fungsi untuk membangkitkan populasi awal secara acak.

- crossover

  Fungsi untuk melakukan proses *order crossover*.

- mutation

  Fungsi untuk melakukan proses *insertion mutation*.

- elitism

  Fungsi untuk membangun populasi baru untuk generasi berikutnya.

- validatePath

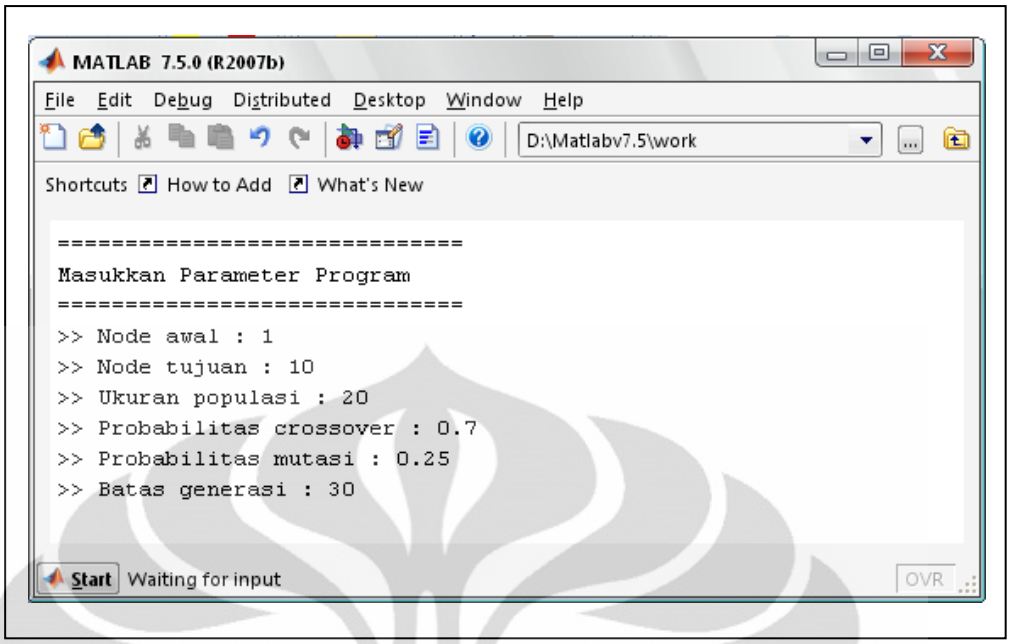
  Fungsi untuk memeriksa apakah jalur yang terkandung dalam kromosom adalah *valid* atau tidak.

Pemanggilan program dilakukan dengan cara mengetik "gaspp" pada MATLAB *Command Window*. Program akan berhenti apabila telah mencapai batas generasi yang telah ditentukan oleh pengguna.

Contoh pemanggilan program saat memasukkan data dan hasil keluaran yang diperoleh pada Contoh 4.1.

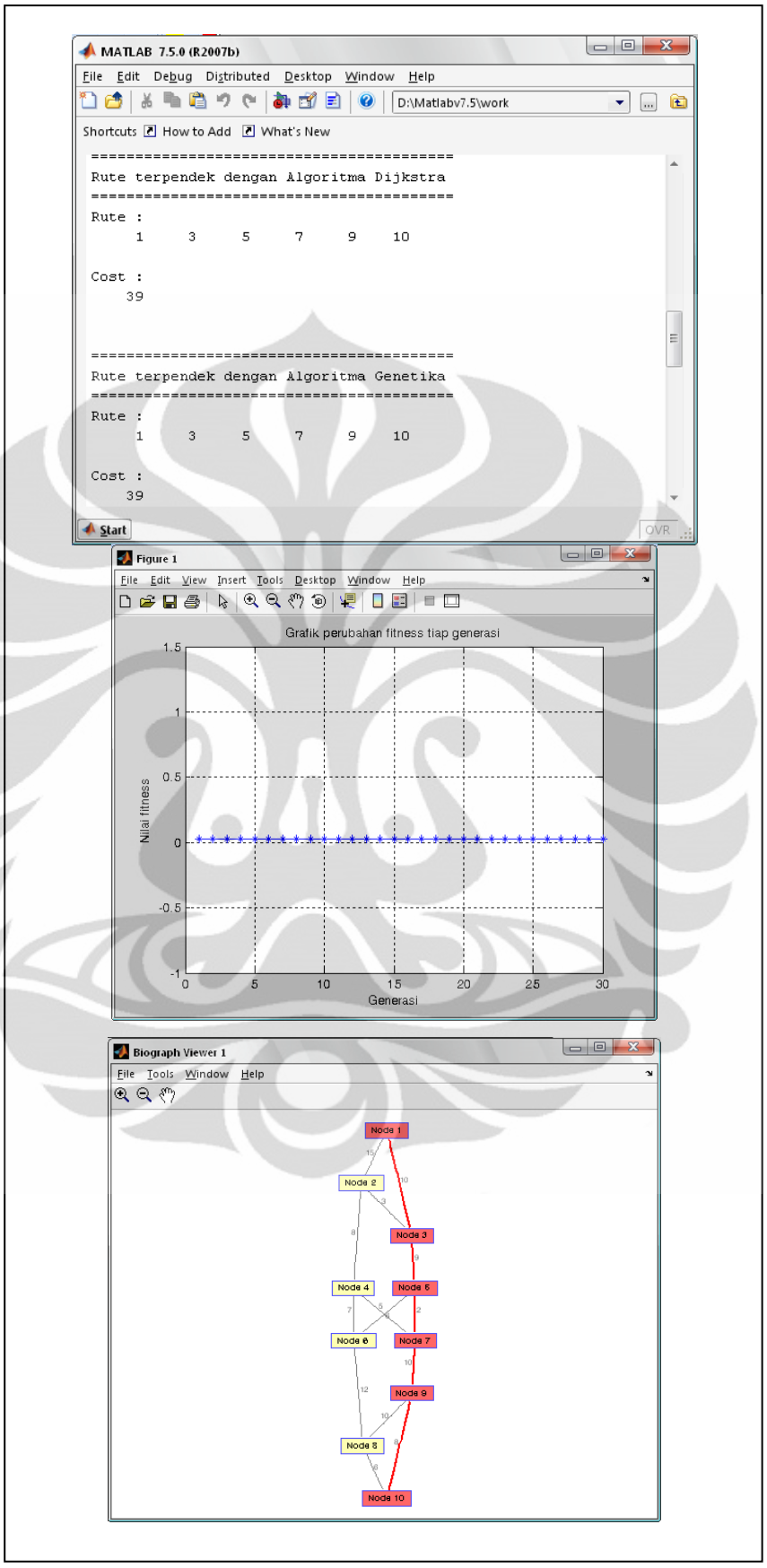
Contoh 4.1 :

Pada contoh ini, masalah pada Contoh 3.1 akan diselesaikan menggunakan algoritma genetika. Ketik "gaspp" pada MATLAB *Command Window* dan tekan *Enter*. Maka program akan meminta pengguna untuk memasukkan *input* berupa *node* sumber, *node* tujuan, dan parameter-parameter algoritma genetika (Gambar 4.1.a).



Gambar 4.1.a Tampilan *command window* data masukan untuk Contoh 4.1

Adapun nilai parameter yang digunakan untuk masalah ini adalah ukuran populasi 20, probabilitas *crossover* 0.7, probabilitas mutasi 0.25, batas generasi 30, sedangkan *node* sumber 1, dan *node* tujuan 10. Program kemudian mengeluarkan solusi akhir yang didapat dan grafik yang menampilkan perubahan nilai *fitness* terbaik terhadap pertambahan generasi (Gambar 4.1.b), serta mencetak solusi dari algoritma Dijkstra yang dapat digunakan sebagai pembanding algoritma genetika oleh pengguna.



Gambar 4.1.b Tampilan keluaran untuk Contoh 4.1

Dari Gambar 4.1.b terlihat bahwa solusi untuk Contoh 3.1 adalah jalur 1-3-5-7-9-10 yang memiliki total *cost* 39. Dari Gambar 4.1.b terlihat pula bahwa solusi optimal dengan menggunakan algoritma Dijkstra menunjukkan rute yang sama, yaitu 1-3-5-7-9-10 dengan total *cost* 39. Grafik pada Gambar 4.1.b menunjukkan nilai *fitness* terbaik pada tiap generasi. Dari grafik pada Gambar 4.1.b, terlihat bahwa solusi optimal telah diperoleh sejak generasi pertama yaitu dengan nilai *fitness* 0.0256.

Selanjutnya akan dibahas beberapa percobaan yang dilakukan untuk melihat pengaruh parameter-parameter genetik seperti, ukuran populasi, probabilitas *crossover*, dan probabilitas mutasi terhadap kinerja algoritma genetika.

4.2 Hasil Percobaan

Percobaan dilakukan dengan mengubah nilai parameter probabilitas *crossover*, probabilitas mutasi, serta ukuran populasi dalam batas generasi yaitu, 50 untuk semua percobaan. Program dijalankan pada *Personal Computer* (PC) dengan prosesor Intel Pentium IV 2,26 GHz, memori 1 GB, dan sistem operasi Microsoft Windows XP Professional Edition.

Masalah pengujian menggunakan dua masalah yang didapat dari *OR-Library* (*OR-Library*, 2011). Kedua masalah pengujian dapat dilihat pada Tabel 4.1. Adapun penentuan keoptimalan solusi dari algoritma genetika akan mengacu kepada solusi dari algoritma Dijkstra, karena algoritma Dijkstra merupakan algoritma terbaik yang masih digunakan sampai saat ini (R. Kumar dan M. Kumar, 2010) untuk penentuan solusi optimal dari masalah jalur terpendek.

Tabel 4.1 Masalah pengujian dari *OR-Library* (*OR-Library*, 2011)

Masalah Pengujian	Banyaknya *Node*	Banyaknya Busur
1	100	955
2	200	2040

4.2.1 Percobaan dengan Mengubah Nilai Parameter Ukuran Populasi (*nind*)

Percobaan pertama dilakukan dengan menggunakan batas generasi = 50, probabilitas *crossover Pc* = 0.75, probabilitas mutasi *Pm* = 0.25, dan ukuran populasi *nind* yang berubah-ubah. Nilai *nind* yang digunakan adalah 10, 50, dan 200. Untuk masalah pengujian 1 dan pengujian 2 masing-masing dilakukan percobaan sebanyak 10 kali. Untuk masalah pengujian 1 didapat solusi optimal dengan algoritma Dijkstra dengan *node* sumber 1 dan *node* tujuan 100 adalah 80, sedangkan untuk masalah pengujian 2 didapat solusi optimal dengan *node* sumber 1 dan *node* tujuan 200 adalah 230. Nilai solusi optimal ini yang akan digunakan untuk menentukan keoptimalan pada percobaan yang akan dilakukan dengan algoritma genetika.

Tabel 4.2 Hasil percobaan untuk masalah pengujian dengan mengubah nilai parameter *nind*

Masalah Pengujian	Banyak Percobaan	*Node* Sumber	*Node* Tujuan	Banyak Solusi Optimal untuk Tiap Ukuran Populasi		
				10	50	200
1	10	1	100	2	4	8
2	10	1	200	0	2	6

Terlihat pada Tabel 4.2 bahwa algoritma genetika untuk percobaan pertama memberikan hasil yang cukup baik. Dengan waktu komputasi rata-rata untuk masalah pengujian 1 berturut-turut berdasarkan ukuran populasinya adalah

0.2 detik, 1.3 detik, dan 5 detik, sedangkan untuk masalah pengujian 2 waktu komputasi rata-rata berturut-turut adalah 0.5 detik, 2.3 detik, dan 10.5 detik. Sementara algoritma Dijkstra untuk masalah pengujian 1 dan masalah pengujian 2 waktu komputasi yang dihabiskan berturut-turut adalah 0.02 detik dan 0.05 detik.

Seperti yang terlihat pada Tabel 4.2 bahwa penambahan nilai parameter *nind* mengakibatkan solusi optimal yang diperoleh menjadi semakin banyak untuk kedua masalah pengujian. Hal ini dikarenakan ukuran populasi menentukan banyaknya kombinasi jalur yang ada dalam satu populasi, semakin besar ukuran populasi maka semakin banyak pula variasi dari kromosom atau kombinasi jalur, sehingga semakin besar pula kemungkinan ditemukannya kromosom dengan nilai *fitness* terbaik global atau jalur dengan total *cost* minimum yang dapat dipertahankan dan bahkan diperbaiki lagi hingga akhir generasi yang pada akhirnya dapat menjadi solusi optimal global. Sehingga dapat dikatakan bahwa penambahan nilai parameter *nind* dapat membuat kinerja algoritma genetika lebih baik dalam hal pencarian solusi optimal. Namun penentuan nilai parameter *nind* ini haruslah proporsional dengan banyaknya simpul pada jaringan, karena jika nilai parameter *nind* relatif terlalu tinggi terhadap banyaknya simpul maka tentu saja akan memakan banyak waktu komputasi. Jika nilai parameter *nind* relatif terlalu rendah terhadap banyaknya simpul maka yang terjadi adalah kemungkinan didapat solusi optimal akan semakin rendah, seperti yang terlihat pada Tabel 4.2 di atas.

Selanjutnya akan dilihat pengaruh perubahan nilai parameter *Pc* terhadap kinerja algoritma genetika.

4.2.2 Percobaan dengan Mengubah Nilai Parameter Probabilitas *Crossover* (*Pc*)

Percobaan kedua dilakukan dengan menggunakan batas generasi = 50, probabilitas mutasi *Pm* = 0.25, ukuran populasi *nind* = 100, dan probabilitas *crossover Pc* yang berubah-ubah. Nilai probabilitas *crossover Pc* yang digunakan adalah 0.3, 0.75, dan 1.0.

Tabel 4.3 Hasil percobaan untuk masalah pengujian dengan mengubah nilai parameter *Pc*

Masalah Pengujian	Banyak Percobaan	*Node* Sumber	*Node* Tujuan	Banyak Solusi Optimal untuk Tiap Probabilitas *Crossover*		
				0.3	0.75	1.0
1	10	1	100	1	4	9
2	10	1	200	0	3	7

Terlihat pada Tabel 4.2 bahwa semakin besar nilai probabilitas *crossover* maka semakin banyak pula solusi optimal yang tercapai dalam 10 kali percobaan. Sehingga dapat dikatakan parameter *Pc* dapat mempengaruhi kinerja algoritma genetika, hal ini dikarenakan semakin besar nilai probabilitas *crossover* maka semakin besar pula peluang melahirkan anak atau *offspring* dari kromosom orang tua yang unggul, sehingga pada akhir generasi akan terlahir *offspring* dengan nilai *fitness* yang terbaik.

Selanjutnya akan diuji pengaruh perubahan nilai probabilitas mutasi *Pm* terhadap kinerja algoritma genetik.

4.2.3 Percobaan dengan Mengubah Nilai Parameter Probabilitas Mutasi (*Pm*)

Percobaan ketiga dilakukan dengan menggunakan batas generasi = 50, probabilitas *crossover Pc* = 0.75, ukuran populasi *nind* = 100, dan probabilitas mutasi *Pm* yang berubah-ubah. Nilai probabilitas mutasi *Pm* yang digunakan adalah 0.01, 0.25, dan 0.5.

Tabel 4.4 Hasil percobaan untuk masalah pengujian dengan mengubah nilai parameter *Pm*

Masalah Pengujian	Banyak Percobaan	*Node* Sumber	*Node* Tujuan	Banyak Solusi Optimal untuk Tiap Probabilitas Mutasi		
				0.01	0.25	0.5
1	10	1	100	1	3	3
2	10	1	200	0	2	3

Terlihat pada Tabel 4.4 bahwa semakin besar nilai probabilitas mutasi maka cenderung semakin banyak solusi optimal yang tercapai dalam 10 kali percobaan untuk kedua masalah pengujian. Sehingga dapat dikatakan parameter *Pm* dapat mempengaruhi kinerja algoritma genetik bahwa proses mutasi dapat mencegah terjadinya optimum lokal. Karena itu meskipun pada kehidupan nyata mutasi memiliki dampak yang negatif, namun pada algoritma genetika ini mutasi harus tetap ada.

BAB 5
PENUTUP

5.1 Kesimpulan

Kesimpulan yang dapat diambil dari skripsi ini adalah :

1. Algoritma genetika dapat digunakan untuk menyelesaikan masalah jalur terpendek.
2. Masalah jalur terpendek dengan jumlah simpul dan lintasan yang semakin banyak akan dapat diselesaikan serta mengarah ke titik optimal bila diimbangi dengan peningkatan ukuran populasi dan parameter lainnya dalam algoritma genetika.

5.2 Saran

Mengingat keterbatasan waktu untuk mengembangkan lebih jauh skripsi ini, maka saran-saran untuk pengembangan skripsi ini adalah :

1. Dapat digunakan jenis-jenis operator algoritma genetika yang lainnya untuk memperoleh hasil yang lebih baik dalam pencarian solusi.
2. Program dapat dikembangkan berbasis *Java* agar dapat digunakan lintas *platform*.
3. Perlu dibuat algoritma tambahan untuk membangkitkan populasi awal, agar dapat menjamin keoptimalan algoritma genetika dengan waktu komputasi yang tidak terlalu banyak.

DAFTAR PUSTAKA

N. K. Cauvery dan K. V. Viswanatha (2009). *Routing in Dynamic Network using Ants and Genetic Algorithm*. International Journal of Computer Science and Network Security, Vol. 9 No.3.

D. A. Lubis (2009). *Implementasi Algoritma Ant Colony System dalam Menentukan Optimisasi Network Routing*. Skripsi: Universitas Sumatera Utara.

D. E. Goldberg (1989). *Genetic Algorithms in search, optimization and machine learning*.

E. Satriyanto (2009). *Algoritma Genetika*. http://lecturer.eepis-its.edu/~kangedi/materi%20kuliah/Kecerdasan%20Buatan/Bab%207%20Algoritma%20Genetika.pdf, Januari 2011.

Foulds (1992). *Graph Theory Applications*. Springer – Verlag, New York.

F. Saptono dan T. Hidayat (2007). *Perancangan Algoritma Genetika untuk Menentukan Jalur Terpendek*. Seminar Nasional Aplikasi Teknologi Informasi. Yogyakarta. 16 Juni 2007.

F. Saptono, I. Mutakhiroh, T. Hidayat, dan A. Fauziyah (2007). *Perbandingan Performansi Algoritma Genetika dan Algoritma Semut untuk Penyelesaian Shortest Path Problem*. Seminar Nasional Sistem dan Informatika. Bali. 16 November 2007.

G. Nagib dan W. G. Ali (2010). *Network Routing Protocol using Genetic Algorithms*. International Journal of Electrical & Computer Sciences, Vol. 10 No. 02.

Internet World Stats (2011). http://www.internetworldstats.com/stats.htm, 04 Juli 2011, pk. 02.08.

K.G. Coman dan A.M. Odlyzko (2001). *Internet growth: Is there a "Moore's Law" for data traffic?* AT&T Labs - Research.

K. Muklis (2004). *Design dan Implementasi Routing Dynamic dengan Algoritma Genetika*. Skripsi: Institut Teknologi Sepuluh Nopember.

M. Obitko (1998). *Introduction to Genetic Algorithm*. University of Applied Sciences.

N. Murniati (2009). *Penerapan Algoritma Genetik pada DNA Sequencing By Hibridization*. Skripsi: Depok, Departemen Matematika, FMIPA, Universitas Indonesia.

OR–Library. http://people.brunel.ac.uk/~mastjjb/jeb/info.html, 22 Mei 2011, pk. 12.00.

R. A. F. Adriansyah (2008). *Analisis Protokol Routing pada Jaringan Komputer Universitas Sumatera Utara dengan Router Simulator*. Skripsi: Universitas Sumatera Utara.

R. Kumar dan M. Kumar (2010). *Exploring Genetic Algorithm for Shortest Path Optimization in Data Networks*. Global Journal of Computer Science and Technology Vol.10.

S. Kusumadewi (2003). *Artificial Intelligence (Teknik dan Aplikasinya)*. Graha Ilmu: Yogyakarta.

S. Lukas, T. Anwar, dan W. Yuliani (2005). *Penerapan Algoritma Genetika untuk Traveling Salesman Problem dengan Menggunakan Metode Order Crossover dan Insertion Mutation*. Seminar Nasional Aplikasi Teknologi Informasi 2005 : I-2.

W. S. E. Tanjung (2010). *Kajian Algoritma Genetika Pada Travelling Salesman Problem*. Skripsi: Universitas Sumatera Utara.

LAMPIRAN 1

Listing Program Algoritma Genetika

- **gaspp**

```
% GENETIC ALGORITHM FOR SHORTEST PATH  PROBLEM  IN DATA NETWORK

function gaspp_v1_4

clear; clc;

% ---------------------------------------------------------------

% PROGRAM INPUT / PROGRAM PARAMETER

% ---------------------------------------------------------------

disp('==============================');
disp('Masukkan Parameter Program');
disp('==============================');
f_input = input('>> Nama Input File : ','s');  % Contains network info
startNode = input('>> Node sumber : ');          % Source node
endNode = input('>> Node tujuan : ');          % Destination node
nind = input('>> Ukuran populasi : ');         % Population size
Pc = input('>> Probabilitas crossover : ');    % Crossover probability
Pm = input('>> Probabilitas mutasi : ');       % Mutation probability
ngener = input('>> Batas generasi : ');        % Generations limit

% ---------------------------------------------------------------

% NETWORK BUILDING

% ---------------------------------------------------------------

fid = fopen(f_input,'r');
num = fscanf(fid, '%d', [1,1]);             % Number of nodes
edges = fscanf(fid, '%d', [1,1]);           % Number of edges
source_node = zeros(1,edges);                % Set of source nodes
dest_node = zeros(1,edges);                 % Set of end nodes
link_cost = zeros(1,edges);                 % Set of link cost
unlink_cost = 1000;                         % Set of unlink cost
for i = 1 : edges
   temp = fscanf(fid, '%d', [1,4]);
   source_node(i) = temp(1);
   dest_node(i) = temp(2);
   link_cost(i) = temp(3);
end
fclose(fid);
net = sparse(source_node, dest_node, link_cost, num, num);
%h = view(biograph(net, [], 'ShowArrows','off', 'ShowWeights','on'));
% Create a matrix table for the costs of each links
matrix_cost = zeros(num);
matrix_cost(:,:) = unlink_cost;
% Create a link matrix of the relationship between the nodes
link_matrix = zeros(num);
for i = 1 : length(link_cost)
   matrix_cost(source_node(i),dest_node(i)) = link_cost(i);
   link_matrix(source_node(i),dest_node(i)) = 1;
   %matrix_cost(dest_node(i),source_node(i)) = link_cost(i);
end
node_distance = dist((rand(num,2))');
for i = 1 : num
   node_distance(i,i) = unlink_cost;
end

disp(' ');
disp('==============================');
disp('Cost untuk masing-masing link');
disp('==============================');
disp(matrix_cost);
```

```
% The shortest path by Dijkstra's algorithm
% ----------------------------------------
disp(' ');
disp('=========================================');
disp('Rute terpendek dengan Algoritma Dijkstra');
disp('=========================================');
tic;
[cost, path] = graphshortestpath(net,startNode,endNode);
toc;
disp('Rute :');
disp(path);
disp('Cost :');
disp(cost);

tic;
% Generate random chromosomes in initial population
% ------------------------------------------------
chrom = initChrom(nind,num,link_matrix,startNode,endNode);

% Fitness evaluation
% ------------------------------------------------
% First calculate the total cost for each chromosomes
ObjV = totalCost(chrom,matrix_cost,nind,endNode,link_matrix);
% The bigger fitness, the better chromosome / path
% It is why the fitness defined as (1/objective value)
fitness = fitnessV(ObjV);

plot_f = zeros(ngener,1);
newchrom = zeros(size(chrom));
numsel = nind;
% ------------------------------------------------------
% START the main iteration of genetic algorithm
% ------------------------------------------------------
for m = 1 : ngener
    % Roulette wheel selection
    % ------------------------------------------------
    % The number of chromosomes to be selected for reproduction
    % (100% of previous population)
    numsel = round(numsel*1);
    % The individual percentage of fitness
    percent = cumsum(fitness/sum(fitness)*100);
    for x = 1 : numsel
       randpoint = 100*rand;
       for y = 1 : nind
          if randpoint <= percent(y)
             newchrom(x,:) = chrom(y,:);
             break
          end
       end
    end
    % end Roulette wheel selection
    % ------------------------------------------------

    % Crossover
    % ------------------------------------------------
    for j = 1 : floor(numsel/2)
       if rand(1) <= Pc
          crosschrom = [newchrom((j*2-1),:);newchrom((j*2),:)];
          temp = crossover(crosschrom);
          % Validate the paths / offspring chromosomes
          valid = validatePath(temp,matrix_cost,unlink_cost,endNode);
          if valid(1,1) == 1
             newchrom((j*2-1),:) = temp(1,:);
          end
          if valid(2,1) == 1
             newchrom((j*2),:) = temp(2,:);
          end
       end
    end
    % end Crossover
    % ------------------------------------------------

    % Mutation
```

```
    % ------------------------------------------------
    for j = 1 : numsel
       if rand(1) <= Pm
          temp = mutation(newchrom(j,:));
          % Validate the path / offspring chromosome
          if validatePath(temp,matrix_cost,unlink_cost,endNode) == 1
             newchrom(j,:) = temp;
          end
       end
    end
    % end Mutation
    % ------------------------------------------------

    % Creating a new population for new generation
    % ------------------------------------------------
    % Preserving a part of the parent chromosome population
    chrom = elitism(fitness,newchrom,chrom,matrix_cost,nind,endNode,link_matrix);

    % Fitness evaluation
    % ------------------------------------------------
    % First calculate the total cost for each chromosomes
    ObjV = totalCost(chrom,matrix_cost,nind,endNode,link_matrix);
    % The bigger fitness, the better chromosome / path
    % It is why the fitness defined as (1/objective value) or (1/cost)
    fitness = fitnessV(ObjV);
    sf = sort(fitness,'descend');
    plot_f(m) = sf(1);

  % What path is the best from the current population:
  % "b" tells the position of path / chromosome in the population
  [a,b] = min(ObjV);

  path = zeros(1,find(chrom(b,:) == endNode));
  for j = 1 : length(path)
     path(j) = chrom(b,j);
  end
  % View the shortest path in the network
  %set(h.Nodes(path),'Color',[1 0.4 0.4])
  %fowEdges = getedgesbynodeid(h,get(h.Nodes(path),'ID'));
  %revEdges = getedgesbynodeid(h,get(h.Nodes(fliplr(path)),'ID'));
  %edges = [fowEdges;revEdges];
  %set(edges,'LineColor',[1 0 0])
  %set(edges,'LineWidth',1.5)
end
% ------------------------------------------------------
% END main iteration
% ------------------------------------------------------
toc;
% The shortest path by Genetic algorithm
% ---------------------------------------
disp(' ');
disp('=======================================');
disp('Rute terpendek dengan Algoritma Genetika');
disp('=======================================');
disp('Rute :');
disp(path);
disp('Cost :');
disp(a);

% PLOT FITNESS
% ------------
plot(plot_f,'-*');
title('Grafik perubahan fitness tiap generasi');
xlabel('Generasi')
ylabel('Nilai fitness')
grid on
end
```

- **initChrom**

```
% ------------------------------------------------------------------
```

```
% Procedure of  Generating The Initial Population
% ----------------------------------------------------------------

function chrom = initChrom (nind, num, link_matrix, startNode, endNode)
% function name   : initChrom
% function input  : 1. The number of chromosomes in population
%                   2. The number of nodes
%                   3. Link Matrix
%                   4. Start / source node
%                   5. End / destination node
% function output : The initial Chromosomes
% Define chromosomes variable as an array
chrom = zeros(nind,num);
for k = 1 : nind
   therest = 1:num;
   therest(startNode) = 0;
   % Set the first gene as the start node
   chrom(k,1) = startNode;
   % Generate the random genes but must represents a valid path
   thelink = startNode;
   for i = 2 : num
      randNode = find(link_matrix(thelink,:));
      if size(randNode,2) == 0
         break
      end
      temp = round(1 + (size(randNode,2)-1) * rand(1));
      % Do not allow cycle path!
      if size(find(chrom(k,:) == randNode(temp)),2) ~= 0
         for j = 1 : size(randNode,2)
            if size(find(chrom(k,:) == randNode(j)),2) == 0
               temp = j;
               break
            end
         end
         if (j == size(randNode,2))
            if size(find(chrom(k,:) == randNode(j)),2) ~= 0
               break
            end
         end
      end
      chrom(k,i) = randNode(temp);
      therest(randNode(temp)) = 0;
      if randNode(temp) == endNode
         break
      end
      thelink = randNode(temp);
   end
   randNode = find(therest);
   chrom(k,((num-size(randNode,2))+1):num) = randNode;
end
end
```

- **fitnessV**

```
% ----------------------------------------------------------------

% Objective to determine fitness

% ----------------------------------------------------------------

function fitnessV = fitnessV (cost)
% function name   : fitness
% function input  : 1. The cost of each chromosomes
% function output : Array of fitness of each chromosomes
fitnessV = zeros(size(cost,1),1);
for i = 1 : size(cost,1)
   if cost(i) == 0
      fitnessV(i) = 0;
   else
      fitnessV(i) = 1/cost(i);
   end
end
end
```

- **totalCost**

```
% ---------------------------------------------------------------
% Objective function / Total cost of the path
% ---------------------------------------------------------------
function ObjV = totalCost (routes, distances, nind, endNode, link_matrix)
% function name   : totalCost
% function input  : 1. The paths / routes from source to destination node
%                   2. The distance to each nodes
%                   3. The number of chromosomes in the population
%                   4. The destination node
%                   5. The adjacency matrix of relationship between nodes
% function output : Array of total cost for each routes / paths
ObjV = zeros(nind,1);
for k = 1 : nind
   cost = 0;
   for i = 1 : (find(routes(k,:)==endNode)-1)
      temp = link_matrix(routes(k,i),routes(k,(i+1)));
      if temp == 1
         cost = cost + distances(routes(k,i),routes(k,(i+1)));
      else
         cost = 0;
         break
      end
   end
   ObjV(k) = cost;
end
end
```

- **crossover**

```
% ---------------------------------------------------------
% Procedure of OX (Order Crossover)
% ---------------------------------------------------------
function newchrom = crossover (chrom)
% function name   : crossover
% function input  : The old chromosomes as parents
% function output : The new chromosomes as childs
ngenes = length(chrom);
newchrom = chrom;

% Step 1. Define the two points randomly for genes position
% ---------------------------------------------------------
% Randomly in [2,ngenes]
lowpoint = round(2 + ((ngenes)-2).*rand);
uppoint = round(2 + ((ngenes)-2).*rand);
while lowpoint == uppoint
   lowpoint = round(2 + ((ngenes)-2).*rand);
   uppoint = round(2 + ((ngenes)-2).*rand);
end
% Validate the points, must increasing
if lowpoint > uppoint
   temp = lowpoint;
   lowpoint = uppoint;
   uppoint = temp;
end

% Step 2. Ordering and deleting
% -----------------------------
for i = 1 : 2
   if i == 1
      x = 2;
   else
      x = 1;
   end
   j = 1;
   for k = (uppoint+1) : ngenes
```

```
      y = 0;
      for m = lowpoint : uppoint
         if chrom(i,m) == chrom(x,k)
            y = 1;
            break
         end
      end
      if y == 0
         temp(j) = chrom(x,k);
         j = j + 1;
      end
   end
   for k = 2 : uppoint
      y = 0;
      for m = lowpoint : uppoint
         if chrom(i,m) == chrom(x,k)
            y = 1;
            break
         end
      end
      if y == 0
         temp(j) = chrom(x,k);
         j = j + 1;
      end
   end
   j = 1;
   for k = (uppoint+1) : ngenes
      newchrom(i,k) = temp(j);
      j = j + 1;
   end
   for k = 2 : (lowpoint-1)
      newchrom(i,k) = temp(j);
      j = j + 1;
   end
end
end
```

- **mutation**

```
% ------------------------------------------------------
% Procedure of Insertion Mutation
% ------------------------------------------------------
function newchrom = mutation (chrom)
% function name   : mutation
% function input  : The chromosome
% function output : The new chromosome as child
newchrom = chrom;
% Random point between 2 and length(chrom)
% to determine which position of gene in chromosome
% must be mutated
randpoint1 = round(2 + ((length(chrom))-2).*rand);
% Create the second random point to insert
% the selected gene in chromosome and produce a child
randpoint2 = round(2 + ((length(chrom))-2).*rand);
% Insert the selected gene
newchrom(randpoint2) = chrom(randpoint1);
% If the first and last random points are same,
% then do nothing
if randpoint1 ~= randpoint2
   if randpoint1 < randpoint2
      newchrom(randpoint1:(randpoint2-1)) = chrom((randpoint1+1):randpoint2);
   else
      newchrom((randpoint2+1):randpoint1) = chrom(randpoint2:(randpoint1-1));
   end
end
end
```

- **elitism**

```
% ------------------------------------------------------------------------

% Procedure of Elitism

% ------------------------------------------------------------------------

function chrom = elitism (fitness, newchrom, oldchrom, matrix_cost, nind, endNode, link_matrix)
% function name   : elitism
% function input  : 1. The fitness of old chromosomes
%                   2. The new chromosomes
%                   3. The previous / old chromosomes
%                   4. The costs
%                   5. The number of population
%                   6. The destination node
%                   7. The adjacency matrix of relationship between nodes
% function output : The chromosomes for the new generation
fitness2 = fitnessV(totalCost(newchrom,matrix_cost,nind,endNode,link_matrix));
chrom = [fitness oldchrom;fitness2 newchrom];
temp = sortrows(chrom);
chrom = temp((size(temp,1)-nind+1):size(temp,1),2:size(temp,2));
end
```

- **validatePath**

```
% ------------------------------------------------------------

% Procedure of Validating The Path(s)

% ------------------------------------------------------------

function valid = validatePath (chrom, distances, unlink_cost, endNode)
% function name   : validatePath
% function input  : 1. The paths / routes that want to validated
%                   2. The distance to each nodes
%                   3. The cost that the link doesnt exist
%                   4. The destination node
% function output : The boolean values. If the path(s) is valid or not
row = size(chrom);
valid = zeros(row(1),1);
valid(:,1) = 1;
for j = 1 : row(1)
   for i = 1 : (find(chrom(j,:)==endNode)-1)
      if distances(chrom(j,i),chrom(j,i+1)) == unlink_cost
         valid(j,1) = 0;
         break
      end
   end
end
end
```

LAMPIRAN 2

Format *File Input*

```
10
14
1 2 15 0
1 3 10 0
2 3 3 0
2 4 8 0
3 5 9 0
4 6 7 0
4 7 5 0
5 6 6 0
5 7 2 0
6 8 12 0
7 9 10 0
8 10 6 0
9 8 10 0
9 10 8 0
```

Baris pertama menyatakan banyaknya simpul. Baris kedua menyatakan banyaknya busur (*link*). Kemudian baris ketiga dan seterusnya menyatakan *node* sumber, *node* tujuan, dan *cost* dari *link* tersebut.