PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK UNTUK PENYELESAIAN TRAVELING SALESMAN PROBLEM DENGAN MENGGUNAKAN ALGORITMA GENETIKA

TUGAS AKHIR

Oleh :

ENTIN MARTIANA KUSUMANINGTYAS

NRP. 2692 100 007

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA

1999

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK UNTUK PENYELESAIAN TRAVELING SALESMAN PROBLEM DENGAN MENGGUNAKAN ALGORITMA GENETIKA

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer
Pada
Jurusan Teknik Informatika
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya

Mengetahui / Menyetujui,

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Ir. ARIF DJUNAIDY, M.Sc.	Ir. SON KUSWADI
NIP. 131 633 403	NIP. 131 793 741

DEPARTEMEN PENDIDIKAN DAN KEBUDAYAAN INSTITUT TEKNOLOGI SEPULUH NOPEMBER FAKULTAS TEKNOLOGI INDUSTRI JURUSAN TEKNIK INFORMATIKA

S U R A B A Y A
Agustus, 1999

PERSEMBAHAN
(DEDICATION)

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

۞ إِنَّ ٱللَّهَ ٱشْتَرَىٰ مِنَ ٱلْمُؤْمِنِينَ أَنفُسَهُمْ وَأَمْوَٰلَهُم بِأَنَّ لَهُمُ ٱلْجَنَّةَ ۚ يُقَٰتِلُونَ فِى سَبِيلِ ٱللَّهِ فَيَقْتُلُونَ وَيُقْتَلُونَ ۖ وَعْدًا عَلَيْهِ حَقًّا فِى ٱلتَّوْرَىٰةِ وَٱلْإِنجِيلِ وَٱلْقُرْءَانِ ۚ وَمَنْ أَوْفَىٰ بِعَهْدِهِۦ مِنَ ٱللَّهِ ۚ فَٱسْتَبْشِرُوا۟ بِبَيْعِكُمُ ٱلَّذِى بَايَعْتُم بِهِۦ ۚ وَذَٰلِكَ هُوَ ٱلْفَوْزُ ٱلْعَظِيمُ

Sesungguhnya Alloh telah membeli dari orang-orang yang beriman
dengan jiwa mereka dan harta mereka dengan balasan berupa Surga.
Mereka berperang di jalan Alloh, lalu mereka membunuh dan terbunuh.
Itu adalah janji yang benar dari Alloh dalam Taurat, Injil serta Al Qur'an.
Dan siapakah yang lebih menepati janjinya selain Alloh.
Oleh karena itu bergembiralah dengan jual-beli yang telah kamu lakukan itu.
Ketahuilah jual-beli yang demikian adalah kemenangan yang besar.

(Al Qur'an Surat At Taubah : 111)

اللهم أنت ربى لا إله إلا أنت خلقتنى وأنا عبدك وأنا على عهدك

و وعدك ما استطعت أعوذبك من شرما صنعت أبوء لك

بنعمتك علي وأبوء بالذ نب فاغفرلي

فإنه لا يغفر الذنوب إلا أنت

Ya ALloh, Engkau adalah Rabbku, tiada Ilah selain Engkau,
Engkau yang menciptakanku dan aku hamba-Mu,
Dan aku berada dalam perjanjian dengan-Mu dan janji-Mu sekuat tenagaku,
Aku berlindung kepada-Mu dari kejahatan perbuatanku,
Aku kembali kepada-Mu dengan segala ni'mat-Mu kepadaku,
Dan aku kembali kepada-Mu dengan membawa dosa-dosaku,
Maka ampunilah aku karena tiada yang mengampuni dosa-dosa selain Engkau.,
Amin Yaa Robbal 'Aalamiin.

"People can survive & strike with their around persons"
In this time, when this Final Project has finished, I dedicate my appreciation for my nearest persons
-I wish ALloh gives the best answer for you-:

❁ **Mr. & Mrs. Soekarno -my beloved parent-,**

I can't tell you how much I appreciate all of the continued love, praying, guidance, patience and support that you have given me over the years.
I dedicate this graduation as one of my devotions.
My happiness and my ambition if I've could make you proud of me,
in the world and in the great beyond.

❁ **My big brothers and my big sisters**
including my brothers and my sister in law,

Thank's for all the years of support and for giving such great care of my study.
Especially for Noenk & Dick, Wishing you the best of luck and Be the good big brother that ALloh wanted, guys !!!

❁ **My nephews and my nieces,**

Don't ever let yourselves lose the right stuff.
I wish you the best and want you to know that you have my love and my support and of course I always there for you!

❁ **Atiek & Umi -my best friends-,**

I'm so proud with what we have created together.
Thank you for being so flexible with me in the moment when we have shared together and for making the big move with me.
Wishing ALloh always define us as the best persons.

❁ **My marvellous sisters,**

I can't thank you enough for sharing your incredible thinking with me.
Thank you so much for making my live so much better, especially for your continued guidance.

❁ **Asmaul Husna,**

Special thank's to you for all the wonderful moments when we had shared together.

P.S.: tell to Rusydah that

I want to be her best aunt that she can ask for.

❁ **All my little sisters who support me,**

You always give me inspirations as a big sister.

Wishing Alloh defines us as a part of Khoiru Ummah. Ameen!

And don't ever let yourselves late to fight it.

❁ **Eri, Marta, Rahmie.**

I cherish our friendship. I'm so happy if I will be able to share more time with you.

❁ Finally I'd like to thank with very special to:

All Peoples who call the ummah to submit

all the orders and the prohibitions of ALloh SWT.

Surabaya August 16th, 1999

Entin Martiana

ABSTRAK

ABSTRAK

Traveling Salesman Problem (TSP) dikenal sebagai suatu permasalahan yang bersifat *Nondeterministic Polynomial-time Complete*, dimana tidak ada penyelesaian yang paling optimal selain harus mencoba seluruh kemungkinan penyelesaian yang ada. Akan tetapi, bilamana hal tersebut dilakukan, akan dibutuhkan waktu komputasi yang tidak sedikit. Oleh karena itu dibutuhkan sebuah metode pencarian heuristik untuk menyelesaikan permasalahan ini. Sementara di satu sisi yang lain, dalam riset di bidang optimasi belum ditemukan satupun metode pencarian yang dapat menjamin ditemukannya nilai optimal dalam menyelesaikan permasalahan ini. Keberadaan Algoritma Genetika, sebagai sebuah metode *adaptive* yang berusaha memecahkan suatu pencarian nilai dalam sebuah masalah optimasi, menjadi salah satu alternatif dalam memecahkan permasalahan ini.

Dalam Tugas Akhir ini, dibuat sebuah perangkat lunak untuk menyelesaikan TSP dengan menggunakan Algoritma Genetika. Proses Algoritma Genetika yang digunakan dirancang sedemikian rupa sehingga dapat menyelesaikan permasalahan secara akurat untuk permasalahan ini. Tahapan yang dilakukan pertama kali adalah melakukan pengkodean yang sesuai untuk TSP. Yang pada gilirannya dari tahapan ini dibangun populasi dan dari sini kemudian dilanjutkan dengan proses-proses reproduksi dan rekombinasi dari Algoritma Genetika. Selain itu untuk menghindari terjadinya konvergensi dini dan dominasi individu-individu tertentu pada suatu waktu, maka dilakukan mekanisme penyesuaian dan pemilihan operator-operator yang tepat untuk proses rekombinasi. Sehingga dari Algoritma Genetika ini, dihasilkan nilai optimum dari TSP berupa jarak lintasan yang paling pendek.

Dengan perangkat lunak ini pengguna bisa mendapatkan nilai optimal suatu rute TSP dengan lintasan yang benar. Perangkat lunak yang dibuat, dapat dikembangkan lebih lanjut pada permasalahan-permasalahan turunan TSP seperti desain jaringan telepon dan integrasi sirkuit.

KATA PENGANTAR

KATA PENGANTAR

BismiLlahirrohmanirrohiim

AlhamduliLlah, hakikat puji hanya milik **ALloh**, yang memiliki apa yang di langit dan di bumi. Hanya kepadaNya-lah segala pujian di akhirat kelak. Dialah **Al-'Aliim** (yang mempunyai segala ilmu) dan **Ar-Raafi'** (yang Maha Meninggikan) yang hanya karena atas limpahan kasih-sayang, berkah dan taufiqNya-lah penulis dapat menyelesaikan Tugas Akhir ini.

Shalawat serta salam semoga tetap atas junjungan kita, Muhammad, Rasulullah, makhluk-Nya yang terbaik, sekaligus penutup para Nabi dan Rasul yang paling mulia.

Tugas Akhir dengan judul **"Perancangan dan Pembuatan Perangkat Lunak untuk Penyelesaian Traveling Salesman Problem dengan Menggunakan Algoritma Genetika"** ini bukanlah sesuatu yang baru. Apa yang penulis lakukan di sini adalah dalam rangka menerapkan ilmu yang selama ini dipelajari di jurusan Teknik Informatika. Pengalaman paling berharga dan berkesan yang penulis dapatkan adalah pengalaman proses. Suatu proses dari tahapan konsep, desain sampai aplikasi penerapan telah penulis lalui dalam mengerjakan Tugas Akhir ini. Yang tidak kalah penting dari semua proses tersebut adalah pengalaman interaksi sosial yang telah penulis lakukan dengan berbagai pihak yang membantu dalam menyelesaikan Tugas Akhir ini.

Untuk itu dengan selesainya Tugas Akhir ini, penulis menyampaikan terima kasih dan penghargaan kepada beberapa pihak yang telah banyak memberikan bantuan baik dalam penyelesaian Tugas Akhir ini maupun pada saat penulis kuliah di Teknik Informatika, antara lain:

1. Bapak Dr. Ir. Arief Djunaidy, M.Sc. sebagai Ketua Jurusan sekaligus sebagai Dosen Pembimbing I atas bimbingan, dorongan, diskusi, kesempatan dan nasehat-nasehat yang telah diberikan. Semoga ALloh membalas dengan kebaikan yang lebih besar.
2. Bapak Ir. Son Kuswadi, yang telah banyak memberikan bimbingan, dorongan, pengertian dan bantuan selama penyelesaian Tugas Akhir kali ini, dalam posisi sebagai Dosen Pembimbing II. Hanya ALloh yang akan membalas dengan kebaikan yang lebih besar.
3. Bapak Ir. F.X. Arunanto, sebagai Dosen Wali selama penulis kuliah di Teknik Informatika ITS.
4. Ibu Ir. Esther Hanaya, M.Sc. yang pernah menjadi Dosen Wali pengganti bagi penulis, yang juga selalu memberikan dorongan untuk menyelesaikan Tugas Akhir.
5. Bapak Agus Zainal, S.Kom. atas bantuan dan dorongan yang diberikan selama penulis kuliah di Teknik Informatika.
6. Bapak Ir. Aris Tjahyanto M.Kom., Bapak Febriliyan, S.Kom., Bapak Dr. Ir. Joko Lianto M.Sc. sebagai dosen penguji atas kesempatan yang diberikan kepada penulis.
7. Bapak dan Ibu Dosen Teknik Informatika lain, yang telah melimpahkan ilmunya kepada penulis. Semoga menjadi ilmu yang berkah dalam kehidupan.
8. Bapak, Ibu Staff Tata Usaha Teknik Informatika (Pak Yudi, Pak Kadir, Mbak Irna, Pak Sugeng, Pak Mu'in), yang senantiasa memberikan pelayanan administrasi untuk membantu kelancaran perkuliahan.

9. Bapak & Ibu Sumaryono, atas segala bantuan dan kemudahan yang telah diberikan kepada penulis. Semoga ALloh memberikan balasan yang terbaik.
10. Ali, Agung F., Aminuddin and Didik, atas segala bantuan, dorongan dan kerjasama, baik selama perkuliahan maupun pada saat pengerjaan Tugas Akhir.
11. Mbak Wieta, Weny, Lizda, atas dorongan yang senantiasa diberikan kepada penulis.
12. Sony N. atas buku-bukunya yang AlhamduliLlah telah membuat Tugas Akhir ini jauh lebih baik.
13. Mas Agus Soekamto & David, atas bantuannya selama Kerja Praktek.
14. Anib & Herlambang, atas bantuannya (hardware & software) selama pengerjaan Tugas Akhir.
15. Panitia Seminar Tugas Akhir Agustus'99: Ratna, Kholifah, Rizky, Kendi, dll. atas bantuan yang diberikan pada saat Seminar dan Ujian Tugas Akhir.
16. Semua teman di C-08 : Dian, Edi Setyawan, Irfan, Indra, Nugroho dan lain-lain yang tidak bisa penulis sebutkan satu-persatu di sini, atas kebersamaan dan kerjasama selama masa perkuliahan.

Tidak ada kesempurnaan dalam sebuah karya manusia. Kami menyadari sebagai manusia -tempatnya salah dan lupa-, masih banyak kekurangan dalam penyusunan Tugas Akhir ini. Jika ada yang benar dan bermanfaat apa yang kami maksud dari Tugas Akhir ini, maka kami memuji syukur kepada ALloh. Dan jika kami membuat kesalahan, maka kami memohon pengampunan dan maaf kepada ALloh. Untuk itu pula jika ada tegur sapa, kritik dari pembaca Insya ALloh akan diterima dengan lapang dada.

Sebuah ilmu akan bermanfaat kalau ilmu tersebut dapat dikembangkan untuk membantu kehidupan manusia. Kemungkinan pengembangan dan perbaikan dari Tugas Akhir ini masih banyak sekali. Penulis berharap Tugas Akhir ini dapat menjadi sarana untuk dilakukannya penelitian berikutnya, sehingga Tugas Akhir ini dapat berguna ataupun dikembangkan untuk membantu kehidupan.

Akhir kata penulis berdo'a semoga ALloh selalu membuka hati kita dengan cahayaNya dan mengajarkan ilmuNya kepada diri kita dan semoga ALloh menghindarkan kita dari ilmu yang tidak bermanfaat. *ALlahummaftah qulubana bi nuuriKa wa 'alamna bi'ilmiKa. Rabbana na'udzubika minal 'ilmi la yanfa.*

WaLlahu 'alam bish-showab

Surabaya, Agustus 1999

Penulis

DAFTAR ISI

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

DAFTAR TABEL

Halaman

BAB I
PENDAHULUAN

BAB I

PENDAHULUAN

1.1 LATAR BELAKANG

Dalam era globalisasi dan informasi saat ini, kebutuhan akan kecepatan dan ketepatan dalam pengolahan data dan informasi menjadi sesuatu yang sangat diperlukan. Terhadap kondisi inilah manusia selalu berusaha mencari cara untuk memenuhinya. Kecerdasan Buatan sebagai salah satu cabang ilmu dalam bidang Informatika menjadi salah satu andalan untuk selalu menciptakan inovasi dalam melakukan hal tersebut.

Pada tahun 1975, John Holland telah berhasil menciptakan Algoritma Genetika, yaitu sebuah metode *adaptive* yang bisa memecahkan suatu pencarian nilai dalam sebuah masalah optimasi. Algoritma ini didasarkan pada proses genetik yang ada pada makhluk hidup. Yaitu perkembangan generasi dalam sebuah populasi yang alami, secara lambat laun mengikuti prinsip seleksi alam atau "siapa yang kuat, dia yang bertahan (*survive*)". Dengan meniru proses ini, Algoritma Genetika mencari solusi untuk permasalahan-permasalahan dunia nyata.

Traveling Salesman Problem adalah permasalahan optimasi yang klasik. Permasalahan ini melibatkan seorang *traveling salesman* yang harus melakukan kunjungan sekali pada semua kota dalam sebuah lintasan sebelum dia kembali ke titik awal, sehingga pada waktu tersebut perjalanannya dikatakan sempurna. Penyelesaian dari permasalahan ini adalah nilai optimum dari rute yang paling murah, yaitu perjalanan dengan jarak terpendek atau yang mempunyai total harga minimum.

TSP adalah sebuah permasalahan yang bersifat NPC (*Nondeterministic Polynomial-time Complete*), dimana permasalahan ini tidak dapat diselesaikan dengan sebuah algoritma linier standar jika kompleksitas problem besar. Suatu permasalahan yang bersifat NPC tidak mempunyai penyelesaian yang lebih baik daripada mencoba semua kemungkinan penyelesaian yang ada. Hanya sayangnya, bila cara tersebut dilakukan maka dibutuhkan waktu komputasi yang tidak sedikit untuk menyelesaikan permasalahan ini. Waktu komputasi akan bertambah seiring dengan bertambahnya suatu faktor dalam permasalahan yang bersifat NPC. Dalam TSP, waktu komputasi akan bertambah sesuai dengan pertambahan jumlah kota, dikarenakan semakin banyak kemungkinan lintasan yang harus diperiksa untuk mencari lintasan minimum. Dengan demikian untuk jumlah kota yang besar, maka sebuah algoritma heuristik dibutuhkan.

Keberadaan Algoritma Genetika sebagai sebuah metode adaptive adalah salah satu alternatif untuk itu. Dengan latar belakang itulah, maka dalam Tugas Akhir kali ini dilakukan perancangan dan pembuatan perangkat lunak untuk penyelesaian Traveling Salesman Problem dengan menggunakan Algoritma Genetika.

1.2 TUJUAN DAN MANFAAT

Pada Tugas Akhir ini dibuat suatu perangkat lunak untuk menyelesaikan TSP dengan menggunakan Algoritma Genetika dengan penanganan terhadap permasalahan yang biasa muncul dalam penerapan Algoritma Genetika seperti optimum pada wilayah lokal atau dominasi individu-individu solusi pada generasi tertentu. Tujuan penelitian kali ini dengan menggunakan Algoritma Genetika (sebagai algoritma heuristik) adalah untuk melihat apakah algoritma ini mampu

menyelesaikan TSP dengan kompleksitas yang tinggi, yang kemudian nanti akan dibandingkan dengan algoritma yang lain. Sementara penanganan terhadap konvergensi dini yang dilakukan dalam penelitian kali ini bertujuan untuk menghindari adanya optimum lokal pada penyelesaian yang dilakukan. Dengan demikian dapat diuji apakah algoritma ini mampu menyelesaikan masalah yang diselesaikan hingga bisa menjadi alternatif sebagai sebuah penyelesaian yang lebih tepat dan lebih optimal.

Penelitian ini merupakan studi banding, apakah Algoritma Genetika merupakan suatu metode pencarian yang handal jika diterapkan pada TSP. Dari penelitian ini, diharapkan hasilnya dapat diterapkan pada beberapa permasalahan nyata yang menggunakan TSP seperti desain jaringan telepon dan integrasi sirkuit.

1.3 PERUMUSAN MASALAH

Jika kita mendalami Algoritma Genetika akan kita dapatkan bahwa bagian terpenting dari Algoritma Genetika adalah pindah silang (*crossover*), mutasi, seleksi (penghapusan populasi) dan evaluasi *fitness*.

Konvergensi dini adalah kesulitan yang banyak dijumpai dalam penerapan Algoritma Genetika, juga dalam sebagian besar algoritma pencarian. Dalam penyelesaian permasalahan kombinatorial (contohnya TSP), konvergensi dini terjadi apabila pindah silang yang dilakukan menjadikan daerah konvergensi berada pada daerah optimum lokal.

Dari sini maka untuk dapat mewujudkan perangkat lunak dengan tujuan seperti yang diharapkan, maka secara garis besar ada tiga permasalahan yang harus dipecahkan, yaitu :

1. Pengkodekan sebuah lintasan yang cocok, sehingga penyelesaian mengarah pada hasil yang tepat.
2. Perancangan operator genetik yang dapat diaplikasikan untuk menjaga konsistensi *building blocks* dan mencegah ketidaktepatan
3. Pencegahan konvergensi dini

1.4 BATASAN MASALAH

Rancangan perangkat lunak ini bisa menyelesaikan TSP dengan maksimum jumlah kota 100.

1.5 METODOLOGI

Langkah-langkah dalam menyusun Tugas Akhir ini adalah sebagai berikut:

1. Studi literatur.

   Sebagai langkah awal, pada tahap ini dilakukan studi literatur mengenai Algoritma Genetika, penanganan konvergensi dini pada Algoritma Genetika, *Traveling Salesman Problem* (TSP), serta pemrograman dalam lingkungan sistem operasi Microsoft Windows.

2. Perumusan masalah dan perumusan penyelesaiannya.

   Sebagai langkah berikutnya pada bagian ini dilakukan pendefinisian permasalahan dan batasan-batasannya dan langkah-langkah penyelesaian, yang mengarah pada perangkat lunak yang akan dirancang.

3. Perancangan perangkat lunak.

   Langkah yang dilakukan pada tahap ini adalah perancangan struktur data dan diagram alir yang terkait dengan struktur Algoritma Genetika yang akan digunakan dalam perangkat lunak yang dibuat .

4. Pembuatan perangkat lunak.

   Dari struktur data yang telah dirancang dalam langkah sebelumnya, pada tahap ini diimplementasikan ke dalam sebuah bahasa pemrograman.

5. Pengujian perangkat lunak dengan beberapa input parameter Algoritma Genetika.

   Setelah selesai tahap-tahap di atas, tahap berikutnya adalah dilakukan pengujian terhadap perangkat lunak yang ada, dengan beberapa masukan parameter dan informasi jarak antar kota yang semuanya merupakan masukan dalam perangkat lunak yang dibuat. Kemudian dari sini dilakukan perbaikan-perbaikan yang mengarah pada hasil yang sesuai dengan tujuan perancangan yang ada.

6. Perbandingan hasil uji coba dengan hasil penyelesaian dari metode lain.

   Dari hasil yang didapatkan dari uji coba perangkat lunak, maka pada tahap ini dilakukan perbandingan. Baik itu dengan hasil dari penelitian yang sudah pernah dilakukan, maupun dengan metode lain yang dibuat bersamaan dengan pembuatan perangkat lunak ini

7. Penulisan naskah Tugas Akhir.

   Bagian ini merupakan tahap akhir dari serangkaian metodologi yang dilakukan. Pada bagian ini dilakukan penulisan dokumentasi dari tahapan konsep, tahapan desain sampai tahap akhir, yaitu tahapan aplikasi penerapan.

1.6 SISTEMATIKA PEMBAHASAN

Untuk mempermudah memahami perancangan dan pembuatan Tugas Akhir ini maka dibuat naskah laporan dengan sistematika pembahasan seperti yang dipaparkan di bawah ini.

Bab I berisi mengenai pendahuluan pembahasan, dijelaskan tentang latar belakang, tujuan dan manfaat pembuatan perangkat lunak, perumusan masalah, batasan permasalahan, metodologi yang digunakan untuk menyelesaikan serta sistematika pembahasan yang ada dalam naskah laporan.

Selanjutnya dalam Bab II dijelaskan mengenai Algoritma Genetika, dimana secara garis besar penjelasan berisi mengenai pengenalan algoritma ini, sejarah munculnya, dasar teori yang ada di dalamnya, proses dalam algoritma ini dan perbandingan dengan algoritma lain.

Dalam bab berikutnya, yaitu Bab III dijelaskan mengenai *Traveling Salesman Problem*, dimana pembahasan dimulai dari TSP itu sendiri yang meliputi definisi TSP dan sejarah dimunculkannya. Kemudian pembahasan dilanjutkan dengan penyelesaian TSP dengan menggunakan Algoritma Genetika yang mengarah pada tujuan yang diharapkan. Dan selanjutnya adalah satu contoh sederhana TSP dan penerapan Algoritma Genetika untuk menyelesaikannya. Yang selanjutnya diperlihatkan juga bagaimana metode lain yaitu metode Branch and Bound menyelesaikannya.

Selanjutnya dalam Bab IV dijelaskan mengenai perancangan dan pembuatan perangkat lunak untuk Tugas Akhir ini. Bab ini dibagi menjadi empat subbab, yaitu perancangan data, perancangan proses, perancangan antarmuka dan implementasinya dalam sebuah bahasa pemrograman.

Bab V menjelaskan hasil uji coba perangkat lunak dan pembahasannya. Uji coba dilakukan pada beberapa input dan parameter yang berbeda. Dimana dari hasil yang didapatkan akan dibandingkan hasil tersebut dengan hasil yang didapat dari metode lain. Selain itu dibahas kesesuaian hasil dikaitkan dengan tujuan yang hendak diraih berdasarkan masukan parameter yang berbeda.

Bab VI merupakan bab terakhir dari pembahasan yang berisi kesimpulan dari tugas akhir ini dan saran untuk pengembangan lebih lanjut.

BAB II
ALGORITMA GENETIKA

BAB II

ALGORITMA GENETIKA

Bab ini membahas konsep dasar yang menunjang Tugas Akhir, yaitu Algoritma Genetika. Pembahasan meliputi pengenalan tentang Algoritma Genetika, sejarah dimunculkannya, teori dasar yang ada di dalamnya dan perbandingannya dengan algoritma pencarian yang lain.

2.1 PENDAHULUAN

Algoritma Genetika sebagai cabang dari Algoritma Evolusi merupakan metode *adaptive* yang biasa digunakan untuk memecahkan suatu pencarian nilai dalam sebuah masalah optimasi. 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 atau "siapa yang kuat, dia yang bertahan (*survive*)". Dengan meniru proses ini, Algoritma Genetika dapat digunakan untuk mencari solusi permasalahan-permasalahan dalam dunia nyata.[1]

[1] D. Beasley, David R. Bull, Ralph R. Martin. An Overview of Genetic Algorithms : Part 1, Fundamentals. *University Computing*, Vol.15(2), p:58-69, 1993.

Peletak prinsip dasar sekaligus pencipta Algoritma Genetika adalah John Holland.[2] Algoritma Genetika menggunakan analogi secara langsung dari kebiasaan yang alami yaitu seleksi alam. Algoritma ini bekerja dengan sebuah populasi yang terdiri dari individu-individu, yang masing-masing individu mempresentasikan sebuah solusi yang mungkin bagi persoalan yang ada. Dalam kaitan ini, individu dilambangkan dengan sebuah nilai *fitness* yang akan digunakan untuk mencari solusi terbaik dari persoalan yang ada.

Pertahanan yang tinggi dari individu memberikan kesempatan untuk melakukan reproduksi melalui perkawinan silang dengan individu yang lain dalam populasi tersebut. Individu baru yang dihasilkan dalam hal ini dinamakan keturunan, yang membawa beberapa sifat dari induknya. Sedangkan individu dalam populasi yang tidak terseleksi dalam reproduksi akan mati dengan sendirinya. Dengan jalan ini, beberapa generasi dengan karakteristik yang bagus akan bermunculan dalam populasi tersebut, untuk kemudian dicampur dan ditukar dengan karakter yang lain. Dengan mengawinkan semakin banyak individu, maka akan semakin banyak kemungkinan terbaik yang dapat diperoleh.

Dengan teori genetika seperti di atas, sebelum Algoritma Genetika dapat dijalankan, maka sebuah kode yang sesuai (representatif) untuk persoalan harus dirancang. Untuk ini maka titik solusi dalam ruang permasalahan dikodekan dalam bentuk kromosom/string yang terdiri atas komponen genetik terkecil yaitu gen. Pemakaian bilangan seperti *integer, floating point* dan abjad sebagai allele (nilai gen) memungkinkan penerapan operator genetika yaitu reproduksi (*reproduction*), pindah silang (*crossover*) dan mutasi (*mutation*) untuk menciptakan himpunan titik-titik solusi. Untuk memeriksa hasil optimasi, kita membutuhkan fungsi fitness, yang

---

[2] Ibid.

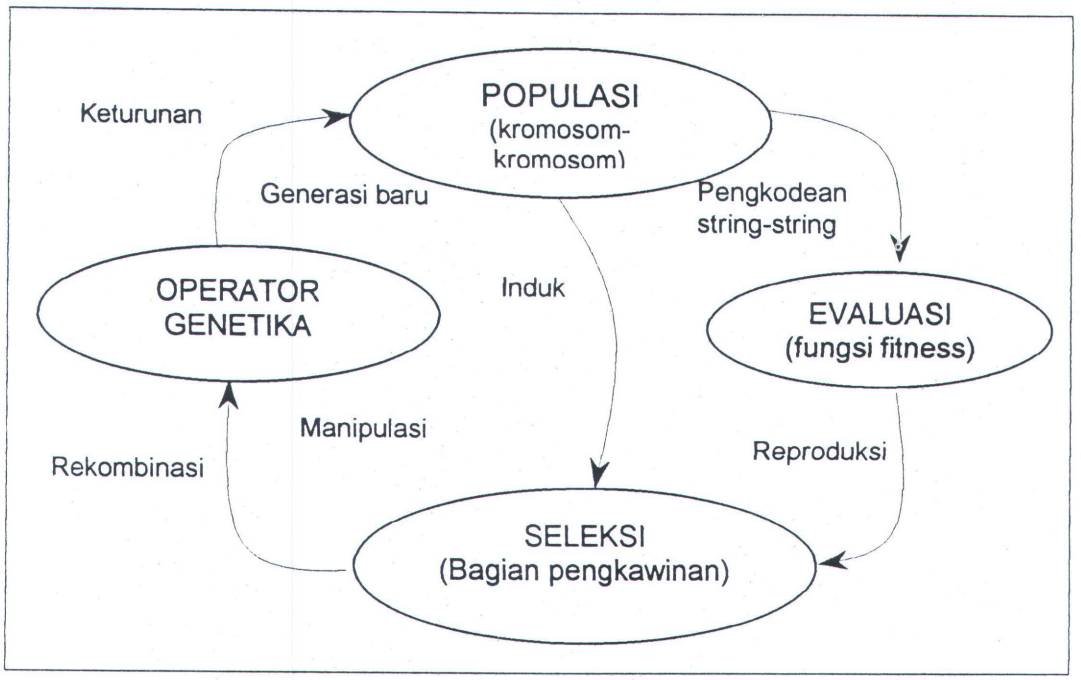
menandakan gambaran hasil (solusi) yang sudah dikodekan. Selama berjalan, induk harus digunakan untuk reproduksi, pindah silang dan mutasi untuk menciptakan keturunan. Jika Algoritma Genetika didesain secara baik, populasi akan mengalami konvergensi dan akan didapatkan sebuah solusi yang optimum.

Secara umum dalam sebuah penerapan Algoritma Genetika akan melalui siklus sederhana yang terdiri dari 4 state, yaitu [3] :

1. Membangun sebuah "populasi" yang terdiri dari beberapa string
2. Evaluasi masing-masing string
3. Proses seleksi agar didapat string yang terbaik
4. Manipulasi genetika untuk menciptakan populasi baru dari string

Ilustrasi mengenai siklus 4 state yang diinspirasikan dari proses biologi untuk proses Algoritma Genetika di atas dapat dilihat pada Gambar 2.1. Setiap siklus yang dilalui dimunculkan generasi baru yang dimungkinkan sebagai solusi bagi permasalahan yang ada.

[3] Jose Ribeno Filho, Cesare Alippi. Philip Treleaven. *Genetic Algorithm Programming Environments.* University College London, White Paper to appear in the IEEE COMPUTER Journal.



Gambar 2.1 **Siklus Algoritma Genetika**

2.2 SEJARAH ALGORITMA GENETIKA

Cikal bakal penggunaan Algoritma Genetika untuk pencarian dalam sistem buatan diprakarsai beberapa ahli Biologi yang menggunakan komputer digital untuk mengerjakan simulasi dari sistem genetika. Di antara para ahli tersebut adalah [4]:

1. Baricelli, N.A. pada tahun 1957 melakukan penemuan *Symbiogenetic Evolution Processes Realised by Artificial Methods*.
2. Baricelli, N.A. pada tahun 1962 melakukan penemuan *Numerical Testing of Evolution Theories*.

[4] Sophia Chan. *Genetic Algorithms, The Origin of Genetic Algorithm*. White Paper, 1996.

3. Fraser, A.S. pada tahun 1960 melakukan penemuan *Simulation of Genetic Systems by Automatic Digital Computers: S-linkage, Dominance, and Epistasis.*
4. Fraser, A.S. pada tahun 1962 melakukan penemuan *Simulation of Genetic Systems.*

Meskipun penelitian-penelitian tersebut bertujuan untuk meneliti gejala alami, namun yang mereka kerjakan tidak begitu jauh dari pemikiran yang memunculkan ide tentang Algoritma Genetika. Fraser mensimulasikan evolusi dari 15-bit biner sebagai string generasi dan menghitung prosentasi dari individu-individu yang terpilih oleh phenotype dengan generasi-generasi yang berurutan. Pada saat itu Fraser tidak menyebutkan dalam laporan-nya bahwa algoritma pencarian dalam gejala alam akan berguna dalam sistem buatan, akan tetapi hasil dari penemuannya menyerupai fungsi optimasi.

Hal itulah yang memberikan inspirasi kepada John Holland dan murid-muridnya untuk mengaplikasikan proses genetika ini pada sistem buatan. Holland menancapkan pondasi terhadap aplikasi ini dengan karya tulisnya dalam teori sistem *adaptive*, diantaranya :

1. *Concerning Efficient Adaptive Systems* (1962)
2. *Information Processing and Processing Systems* (1962)
3. *Outline for A Logical Theory of Adaptive Systems* (1962)

Pada tahun 1962 - 1965 Holland mengikuti kursus-kursus dalam masalah sistem *adaptive* di Universitas Michigan. Salah satu kursusnya adalah *Theory of Adaptive Systems*. Dalam seminar-seminarnya Holland dan murid-muridnya menyempurnakan detail dari Algoritma Genetika dan mengeksperimenkan dengan

parameter-parameternya, terutama menciptakan rumus standar dari Algoritma Genetika.

Selanjutnya teori mengenai Algoritma Genetika ini ditulis oleh John Holland dalam buku karyanya *Adaptation in Natural and Artificial Systems* yang dipublikasikan pada tahun 1975.

Penemuan Holland, Algoritma Genetika ini adalah untuk membuktikan dua hal. Yang pertama, bahwa teori evolusi dapat digunakan sebagai sesuatu yang berguna untuk mencari fungsi-fungsi optimasi pada sebuah komputer. Yang kedua, menciptakan lingkungan kerja bagi teori evolusi dengan sebuah permasalahan dunia nyata.[5]

Dari sinilah Algoritma Genetika diterapkan dan dikembangkan oleh murid-murid Holland. Salah satu muridnya menggunakan Algoritma Genetika pada aplikasi simulasi untuk mencari stategi yang paling mudah pada permainan catur. Murid yang lainnya membuat sebuah disertasi yang menggunakan Algoritma Genetika untuk simulasi fungsi-fungsi dari organisme sel tunggal, sebagai simulasi Biologi pertama yang menggunakan Algoritma Genetika. Ilmuwan-ilmuwan yang selanjutnya mengembangkan Algoritma Genetika adalah D.E. Goldberg, K. De Jong, J.J. Grefenstette, L. Davis, Muhleinbein dan inspirator-inspirator lainnya yang ikut menjadikan Algoritma Genetika berkembang.

---

[5] Sophia Chan. *Genetic Algorithms, John Holland and The Invention of Genetic Algorithms.* White Paper, 1996.

2.3 TEORI DASAR ALGORITMA GENETIKA

Sebelum kita menggunakan Algoritma Genetika, maka kita perlu memahami dulu beberapa teori dasar yang menunjang penerapan algoritma ini. Beberapa teori tersebut adalah sebagai berikut.

2.3.1 Istilah-Istilah dalam Algoritma Genetika

Seperti yang sudah dijelaskan sebelumnya, bahwa algoritma ini berlandaskan pada mekanisme genetika yang ada pada proses alami dan sistem buatan. Sehingga istilah-istilah yang digunakan pada algoritma ini adalah gabungan dari dua disiplin ilmu, yaitu ilmu Biologi dan ilmu Komputer. Tabel 2.1 menjelaskan mengenai istilah-istilah yang digunakan dalam Algoritma Genetika.

Tabel 2.1[6] Penjelasan Istilah dalam Algoritma Genetika

Istilah Biologi yang Digunakan dalam Algoritma Genetika	Keterangan
Kromosom	Individu berupa segmen string yang sudah ditentukan
Gen	Bagian dari string
Loci	Posisi dari gen
Allele	Nilai yang dimasukkan dalam gen
Phenotype	String yang merupakan solusi akhir
Genotype	Sejumlah string hasil perkawinan yang berpotensi sebagai solusi

[6] Mitsuo Gen, Runwei Cheng. *Genetic Algorithms and Engineering Design.* John Wiley & Sons, Inc., New York, 1997, hal. 7.

2.3.2 Parameter Algoritma Genetika

Dalam penerapan Algoritma Genetika, ada beberapa parameter yang dilibatkan, dimana parameter ini digunakan untuk melihat kompleksitas dari algoritma. Jenis parameter yang digunakan bergantung pada permasalahan yang diselesaikan, namun ada beberapa parameter yang menjadi standar, yaitu:

1. Ukuran populasi (*pop_size*).

   Ukuran populasi mempengaruhi unjuk kerja yang baik dan keefektifan Algoritma Genetika. Algoritma Genetika dengan populasi yang kecil, biasanya unjuk kerjanya buruk karena populasi tidak menyediakan cukup materi untuk mencakup ruang persoalan. Populasi yang lebih besar dibutuhkan untuk mempresentasikan keseluruhan ruang persoalan. Dan lagi dengan populasi yang besar dapat mencegah terjadinya konvergensi pada wilayah lokal.

2. Probabilitas pindah silang (p_c).

   Frekuensi operator pindah silang dikendalikan oleh nilai p_c. Dalam setiap populasi, sebanyak p_c * *pop_size* struktur melakukan pindah silang. Semakin tinggi nilai probabilitas pindah silang, semakin cepat struktur baru diperkenalkan dalam populasi. Jika probabilitas pindah silang terlalu tinggi, struktur dengan unjuk kerja yang baik dapat hilang dengan lebih cepat dari seleksi, sehingga populasi tidak bisa meningkatkan unjuk kerja lagi. Sebaliknya, probabilitas yang rendah akan menghalangi proses pencarian.

3. Probabilitas mutasi (p_m).

   Mutasi digunakan untuk meningkatkan variasi populasi, yang mana mutasi ini dilakukan secara acak, tiap unit dasar (bit, posisi, atau token) dalam struktur mempunyai kemungkinan tertentu untuk dipertukarkan. P_m yang rendah dapat mengakibatkan gen-gen yang berpotensi tidak dicoba. Sebaliknya, tingkat

mutasi yang tinggi dapat menyebabkan keturunan kehilangan kemiripan dengan induknya. Diperkirakan terjadinya mutasi sebanyak p_m * *pop_size* * L pada tiap generasi, di mana L adalah panjang struktur dalam setiap individu.

2.3.3 Teorema Skema

Teorema skema merupakan dasar teori yang menjelaskan bagaimana Algoritma Genetika bekerja. Skema adalah sebuah keserupaan pola dalam mendeskripsikan suatu himpunan bagian dari beberapa string yang mempunyai kesamaan pada posisi tertentu. Sebuah skema dibentuk dengan menambahkan sebuah simbol spesial, yaitu simbol ★ (don't care) dalam representasi biner {0,1}. Mekanisme pencocokan skema adalah pada tiap lokasi dari skema alphabet 1 sesuai dengan 1, 0 sesuai dengan 0, dan ★ sesuai dengan 1 dan 0. Sebagai contoh, sebuah skema ★0000 adalah himpunan yang beranggotakan 2 string yaitu {10000, 00000}, juga untuk skema ★111★ mendeskripsikan sebuah himpunan dengan 4 anggota , yaitu {01110, 01111, 11110, 11111}.

Jumlah seluruh string yang dibentuk oleh tiap skema adalah sebanyak 2^r, di mana r adalah jumlah simbol ★ (*don't care*) dalam skema. Dengan kata lain, masing-masing string dengan panjang m akan sesuai dengan 2^m skemata. Sebagai contoh sebuah string (1001110001), akan sesuai dengan 2^{10} skemata:

(1 0 0 1 1 1 0 0 0 1)
(★ 0 0 1 1 1 0 0 0 1)
(1 ★ 0 1 1 1 0 0 0 1)
⋮
(1 0 0 1 1 1 0 0 0 ★)
(★ ★ 0 1 1 1 0 0 0 1)
(★ 0 ★ 1 1 1 0 0 0 1)
⋮
(1 0 0 1 1 1 0 0 ★ ★)
⋮
(★ ★ ★ 1 1 1 0 0 0 1)
⋮
(★ ★ ★ ★ ★ ★ ★ ★ ★ ★)

Jumlah seluruh skemata yang dibentuk pada himpunan string biner dengan panjang = 5, adalah 3*3*3*3*3 = 3^5 = 243. Secara umum, jika kardinalitas alfabet (jumlah huruf dalam alfabet) adalah k dan panjang string adalah L, maka dapat dibentuk $(k+1)^L$ skemata.

Order dari sebuah skema *S* (dinotasikan dengan *o*(*S*)) menunjukkan jumlah dari posisi angka 0 dan 1 yang sudah tetap (bukan posisi *don't care*) yang ada dalam skema. Order ini menunjukkan spesialisasi dari sebuah skema. Sebagai contoh, 3 skemata di bawah ini dengan panjang 10,

S_1 = (★ ★ ★ 0 0 1 ★ 1 1 0)

S_2 = (★ ★ ★ ★ 0 0 ★ ★ 0 ★)

S_3 = (1 1 1 0 1 ★ ★ 0 0 1)

mempunyai order sebagai berikut:

$o(S_1) = 6$, $o(S_2) = 3$, $o(S_3) = 8$,

dan skema S_3 adalah yang paling spesifik.

Angka yang ditunjuk oleh order dari sebuah skema sangat penting untuk menghitung probabilitas dari skema melakukan mutasi (dibahas dalam subbab 2.4.2.4).

Batasan panjang dari skema S (dinotasikan dengan $\delta(S)$) adalah jarak antara posisi angka 0 dan 1 tetap yang pertama dan yang terakhir. Angka ini menunjukkan kerapatan informasi yang ada dalam sebuah skema. Sebagai contoh:

$\delta(S_1) = 10 - 4 = 6$, $\delta(S_2) = 9 - 5 = 4$, $\delta(S_3) = 10 - 1 = 9$.

Angka yang ditunjuk oleh batasan panjang dari sebuah skema akan berguna untuk menghitung probabilitas dari skema untuk melakukan pindah silang (dibahas dalam subbab 2.4.2.3). [7]

Dengan cara ini, maka skemata (jamak dari skema) merupakan suatu cara yang ampuh dan kompak untuk mendefinisikan keserupaan di antara string dengan panjang tetap dan dapat dibentuk dari alphabet tertentu. Yang perlu diperhatikan adalah simbol ★ merupakan meta simbol (sebuah simbol dari simbol-simbol lain). Simbol ini tidak diolah oleh Algoritma Genetika.

2.3.4 Mekanisme Algoritma Genetika

Mekanisme yang ada dalam Algoritma Genetika sangat sederhana, yaitu hanya melibatkan penyalinan string dan pertukaran bagian string. Siklus

---

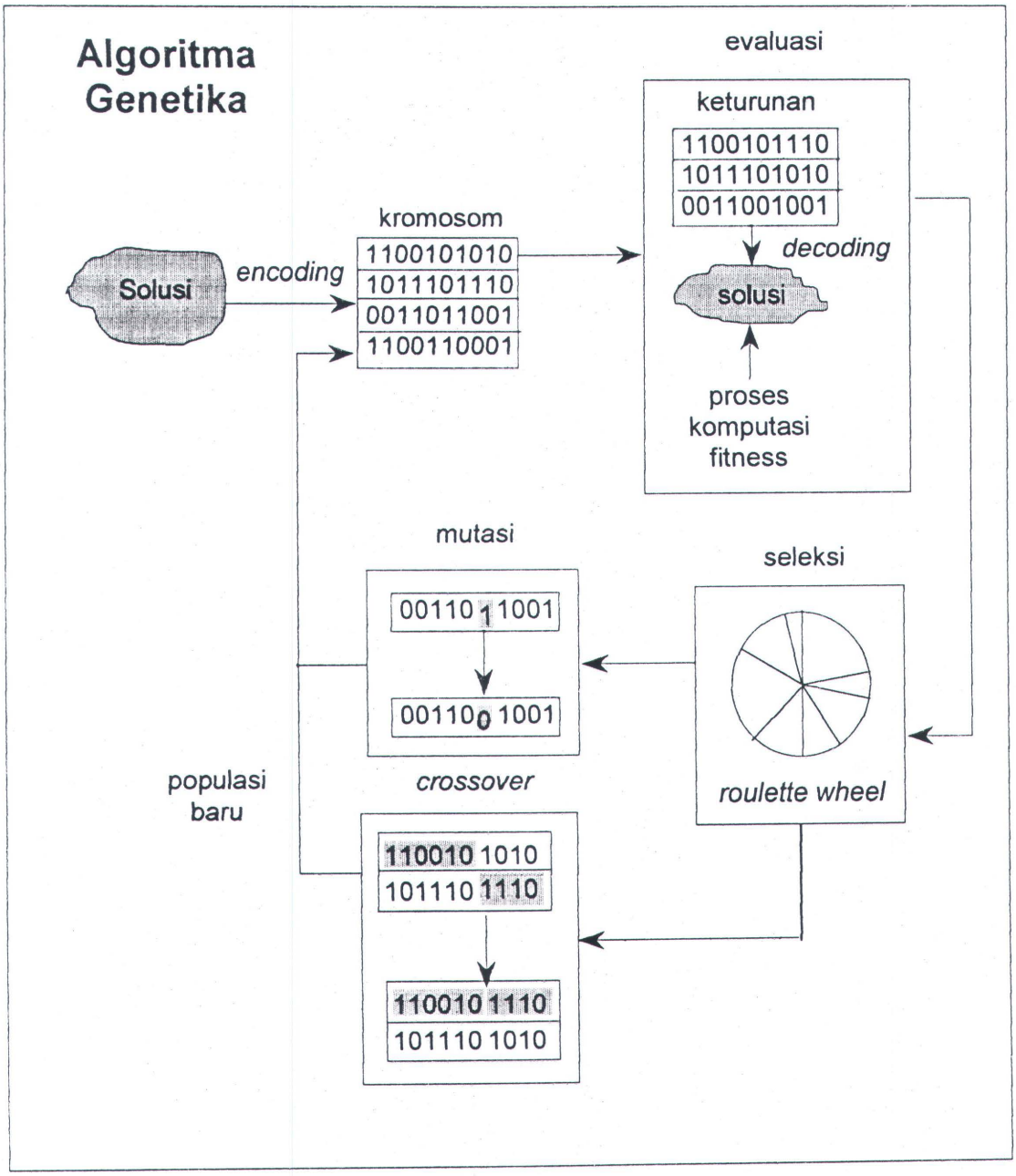
[7] Zbigniew Michalewicz. *Genetic Algorithms + Data Structure = Evolution Programs.* Third Revised and Extended Edition, Springer-Verlag, New York, 1996, hal. 45.

perkembangbiakan Algoritma Genetika diawali dengan pembuatan himpunan solusi secara random dinamakan populasi, dimana di dalamnya terdapat individu-individu yang dinamakan kromosom. Kromosom ini secara lambat laun mengalami iterasi pemilihan dalam sebuah generasi. Selama dalam sebuah generasi, kromosom-kromosom ini dievaluasi, dengan menggunakan rumus-rumus yang ada dalam fungsi fitness. Untuk menciptakan generasi berikutnya dengan kromosom yang baru (dinamakan keturunan) dapat dilakukan dengan menggabungkan dua kromosom yang telah didapat sebelumnya dengan menggunakan operator pindah silang (*crossover*) ataupun dengan memodifikasi sebuah kromosom dengan menggunakan operator mutasi. Sebuah generasi baru sebelum dievaluasi lagi, maka dia melalui proses seleksi berdasarkan fungsi fitness-nya. Dari seleksi ini, kromosom-kromosom yang paling fit mempunyai kemungkinan besar untuk terseleksi.

Setelah beberapa generasi, algoritma akan mengalami konvergen pada kromosom terbaik, yang merupakan nilai optimum dari permasalahan yang diselesaikan. Gambar 2.2 memberikan gambaran mengenai stuktur secara umum mengenai mekanisme perkembangbiakan dalam Algoritma Genetika.

2.4 PROSES ALGORITMA GENETIKA

Hal yang sangat perlu untuk diketahui mengenai proses dalam Algoritma Genetika. Di bawah ini diuraikan mengenai hal tersebut, dimana uraian ini merupakan penjabaran dari mekanisme Algoritma Genetika seperti yang telah dijelaskan pada subbab sebelumnya.



Gambar 2.2 [8]

Ilustrasi Struktur Algoritma Genetika Secara Umum

[8] Op.cit., hal. 3.

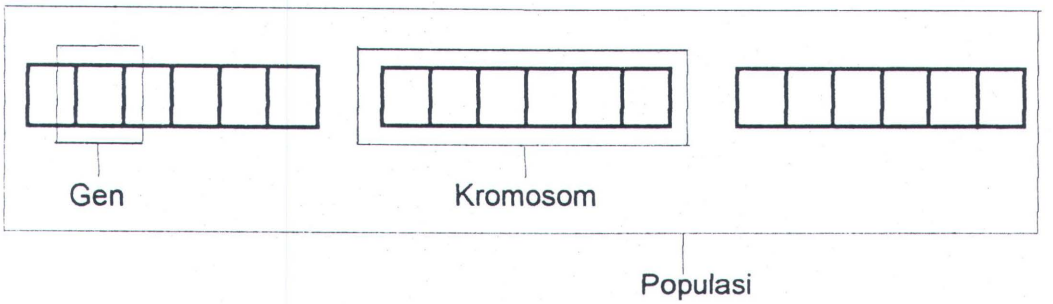
2.4.1 Pengkodean

Langkah pertama kali yang dilakukan dalam penggunaan Algoritma Genetika adalah melakukan pengkodean terhadap permasalahan yang diselesaikan.

Pada algoritma ini dalam mengasumsikan sebuah solusi untuk sebuah persoalan dimungkinkan dengan diwakili satu set parameter. Parameter-parameter ini dinamakan gen berisi nilai-nilai/allele (representasi) yang bersatu membentuk string (kromosom).

Selanjutnya beberapa kromosom yang sejenis berkumpul membentuk populasi. Dari sebuah populasi inilah Algoritma Genetika memulai untuk melakukan pencarian. Ilustrasi mengenai pengkodean ini dapat dilihat pada Gambar 2.3.

Dalam Algoritma Genetika, sebuah fungsi fitness harus dirancang untuk masing-masing masalah yang diselesaikan. Dengan memberikan kromosom tertentu, fungsi fitness menjadikannya salah satu calon fitness atau hasil. Kondisi fitness dari sebuah individu ditentukan oleh penampilan dari phenotype, atau dapat dikatakan bahwa phenotype merupakan kromosom yang menunjukkan hasil. Dari sini dapat disimpulkan bahwa dari genotype berupa kromosom dapat dibentuk phenotype dengan menggunakan fungsi fitness.

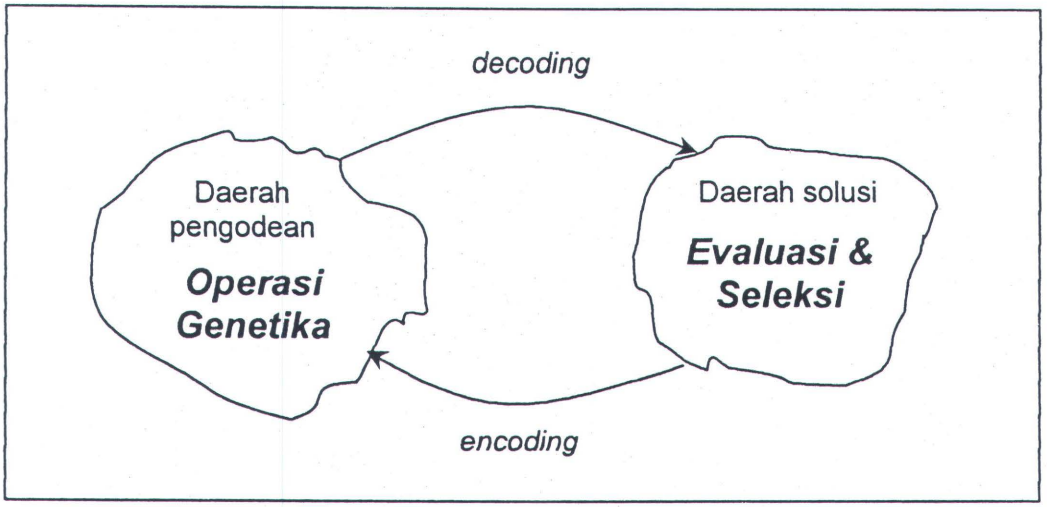


Gambar 2.3 **Pengkodean dalam Algoritma Genetika**

Sebagai contoh untuk pengkodean jika kita mempunyai masalah yaitu mencari maksimum sebuah fungsi dari 3 variabel F(x,y,z) dan akan menampilkan masing-masing variabel dengan 6 bit. Dari contoh ini kita melakukan pengkodean dengan membentuk kromosom yang terdiri dari 3 gen, yang masing-masing gen terdiri dari 6 bit. Sehingga sebuah kromosom terdiri dari 18 bit.

Suatu hal yang mendasar dalam Algoritma Genetika bahwa algoritma ini bekerja pada daerah pengkodean dan daerah solusi. Operasi genetika (pindah silang dan mutasi) bekerja pada daerah pengkodean, sedang proses evaluasi dan proses seleksi bekerja pada daerah solusi. Gambar 2.4 adalah ilustrasi dari kedua daerah ini.

Representasi dalam kromosom merupakan bagian yang penting dalam pengkodean. Di dalam pekerjaannya Holland melakukan representasi pengkodean dengan menggunakan string biner. Dalam berbagai aplikasi Algoritma Genetika, khususnya dalam bidang teknologi industri, representasi yang sederhana ini sulit untuk diaplikasikan langsung, karena string biner bukanlah satu-satunya representasi yang alami. Selama beberapa dekade terakhir, berbagai representasi non biner telah diciptakan untuk berbagai permasalahan khusus. Diantaranya adalah representasi dengan bilangan *real* untuk beberapa permasalahan optimasi berkendala, representasi *integer* untuk permasalahan optimasi kombinatorial.



Gambar 2.4 Daerah Pengkodean dan Daerah Solusi

Dalam hal ini, Goldberg (1989) berpendapat representasi dengan string biner mempunyai jumlah skema (mengenai skema dibahas dalam subbab 2.3.3) yang paling besar. Antonisse (1992) berpendapat bahwa representasi dengan integer mempunyai jumlah skema yang lebih banyak. Janikow dan Michalewicz (1991) membandingkan representasi *biner* dan *floating point*, ternyata representasi dengan *floating point* memberikan hasil yang lebih cepat, lebih konsisten dan lebih akurat.[9]

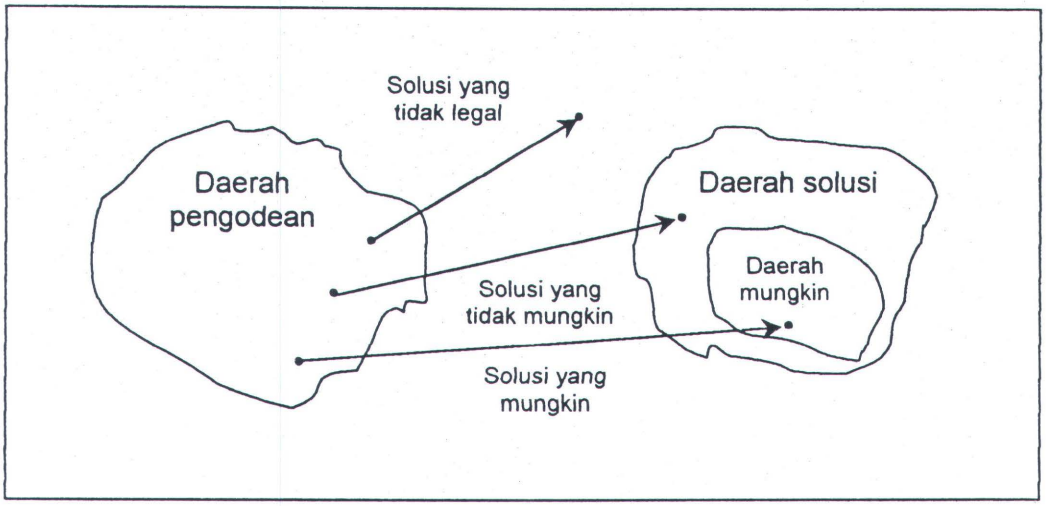
Dalam representasi *non-biner*, ada dua hal yang perlu diperhatikan untuk pengkodean baik dalam phenotype dan genotype, yaitu:

- Feasibilitas dan legalitas dari sebuah kromosom
- Pemetaan

Feasibilitas mempertanyakan apakah sebuah solusi yang direpresentasikan terletak dalam daerah yang *feasible* dari domain permasalahan. Legalitas merujuk pada kenyataan apakah sebuah kromosom benar-benar merepresentasikan

[9] Ibid., hal.16.

sebuah solusi dari domain permasalahan. Dalam Gambar 2.5 di bawah ini diberikan gambaran mengenai feasibilitas dan legalitas dari sebuah pengkodean.



Gambar 2.5 ***Feasibilitas*** **dan** ***Legalitas***

Infeasibilitas kromosom menghilangkan kealamian batasan-batasan dalam permasalahan optimasi. Semua metode, konvensional atau Algoritma Genetika harus memegang batasan ini. Untuk beberapa permasalahan optimasi, daerah yang mungkin dapat direpresentasikan dengan sebuah sistem *equalities* atau *non equalities* (linier atau non linier). Untuk beberapa kasus, metode penalty telah diajukan untuk mengatasi kromosom yang tidak mungkin. Dalam permasalahan optimasi, nilai optimum secara *tipical* akan berada di wilayah antara daerah yang mungkin ataupun yang tidak mungkin. Penerapan metode penalty akan mengeksploitasi pencarian Algoritma Genetika pada nilai optimum di dua daerah tersebut.

Kromosom-kromosom yang tidak legal menghilangkan kealamian dari teknik *encoding*. Untuk beberapa permasalahan kombinatorial, beberapa cara digunakan untuk menghindari keturunan yang tidak legal, karena keturunan yang

tidak legal tidak dapat mewakili sebuah solusi, dalam artian kromosom tersebut tidak dapat dievaluasi. Salah satu cara akan diperkenalkan dalam Bab III adalah PMX operator. Metode ini digunakan untuk mencegah dihasilkannya kromosom yang tidak mungkin dan tidak legal.

Pemetaan dari kromosom-kromosom ke solusi memungkinkan satu dari tiga kasus ini terjadi:

- Pemetaan 1-ke-1
- Pemetaan n-ke-1
- Pemetaan 1-ke-n

Pemetaan satu-satu merupakan pemetaan yang terbaik, karena dari pemetaan ini satu kromosom dapat dipasangkan dengan satu solusi. Pada pemetaan n-satu, sejumlah n kromosom mewakili satu solusi yang sama. Sedangkan pemetaan 1-ke-n adalah pemetaan yang tidak diinginkan. Hal ini dikarenakan pemetaan 1-ke-n merelasikan satu kromosom dengan beberapa solusi, yang mana menyebabkan ketidakjelasan solusi mana sebenarnya yang dirujuk. Pada Gambar 2.6 diberikan gambaran mengenai kemungkinan pemetaan dari daerah pengkodean ke daerah solusi dalam sebuah Algoritma Genetika.

2.4.2 Operasi dalam Algoritma Genetika

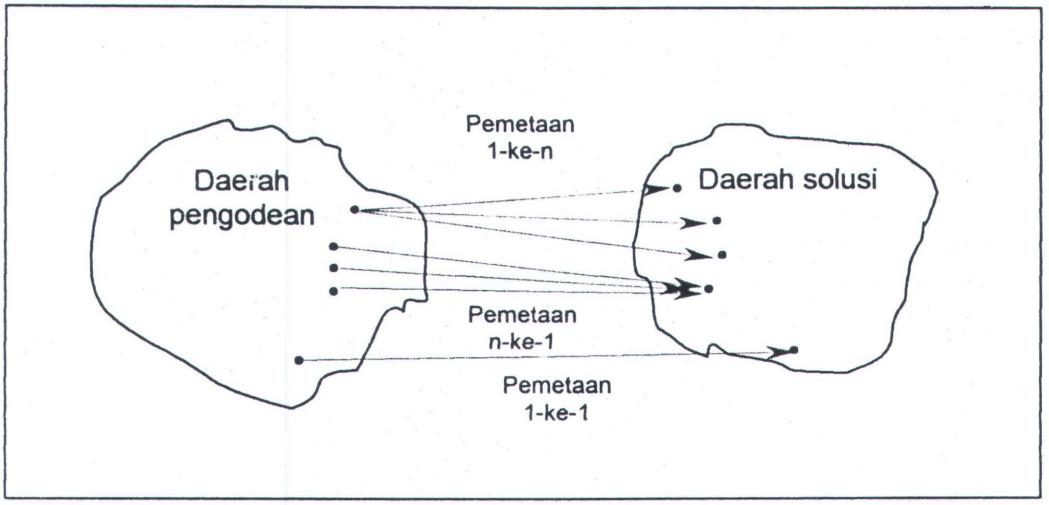
Dengan mekanisme Algoritma Genetika seperti yang dijelaskan pada subbab 2.3.4, setelah dilakukan pendefinisian terhadap kode Algoritma Genetika pada permasalahan yang diselesaikan selanjutnya dilakukan operasi-operasi di bawah ini yang melibatkan 3 operator utama[10] :

1. Operasi Evolusi yang melibatkan proses seleksi (*selection*) di dalamnya.

---

[10] Ibid., hal. 2.

2. Operasi Genetika yang melibatkan operator pindah silang (*crossover*) dan mutasi (*mutation*).



Gambar 2.6 Pemetaan dari kromosom ke solusi

2.4.2.1 Fungsi Evaluasi (Fungsi Fitness)

Disamping representasi, fungsi evaluasi juga merupakan masalah yang penting dalam Algoritma Genetika. Fungsi evaluasi yang baik harus mampu memberikan nilai fitness yang sesuai dengan kinerja kromosom.

Pada permulaan optimasi, biasanya nilai fitness masing-masing individu masih mempunyai rentang yang lebar. Seiring dengan bertambahnya generasi beberapa kromosom mendominasi populasi dan mengakibatkan rentang nilai fitness semakin kecil. Hal ini dapat mengakibatkan konvergensi dini (*premature convergen*).

Problem klasik dalam Algoritma Genetika adalah beberapa kromosom dengan nilai fitness yang tinggi (tetapi bukan nilai optimum) mendominasi populasi dan mengakibatkan Algoritma Genetika konvergen pada *local optima*. Ketika

populasi konvergen, kemampuan Algoritma Genetika untuk mencari solusi yang lebih baik hilang. Tukar silang antara kromosom yang hampir identik menghasilkan offspring yang identik. Dalam kondisi ini hanya operasi mutasi yang mampu menghasilkan kromosom yang relatif baru dan merupakan cara untuk menghindarkan kromosom yang super mendominasi populasi.

2.4.2.2 Seleksi (*Selection*)[11]

Pada proses evolusi Algoritma Genetika, keragaman populasi dan tekanan seleksi (*selection pressure*) memegang peranan penting. Keduanya sangat berkaitan erat. Meningkatnya tekanan seleksi akan berakibat pada minimnya keragaman populasi. Sebaliknya tekanan seleksi yang terlalu longgar membuat proses pencarian menjadi tidak efisien.

Seleksi merupakan proses yang bertanggung jawab atas pemilihan kromosom dalam proses reproduksi. Ada 3 hal yang harus diperhatikan dalam proses seleksi:

- Daerah sampling
- Mekanisme seleksi
- Probabilitas seleksi

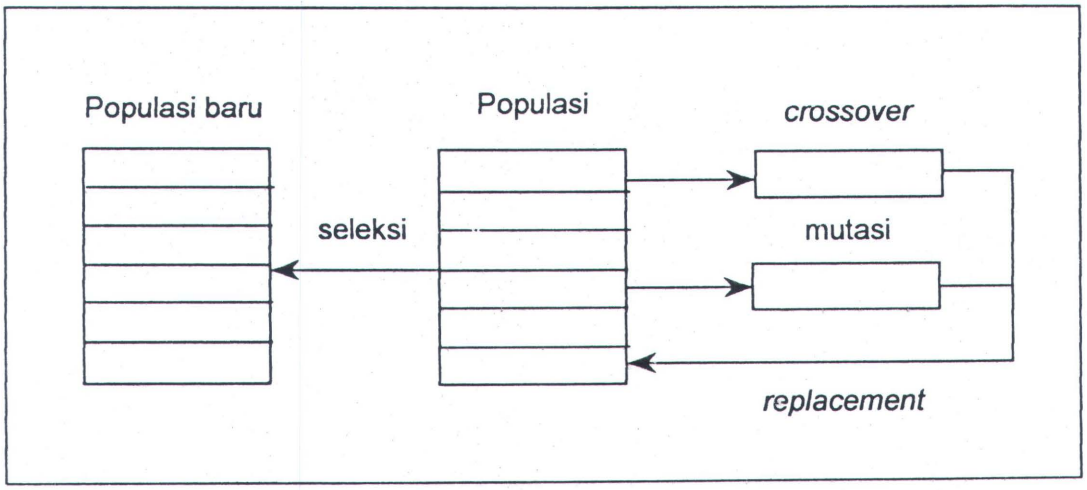
Proses seleksi menghasilkan generasi selanjutnya. Kromosom-kromosom pada generasi selanjutnya mungkin berasal dari semua induk dan semua keturunan atau sebagian dari keduanya. Persoalan asal kromosom ini disebut dengan persoalan daerah sampling.

---

[11] Ibid., hal. 20.

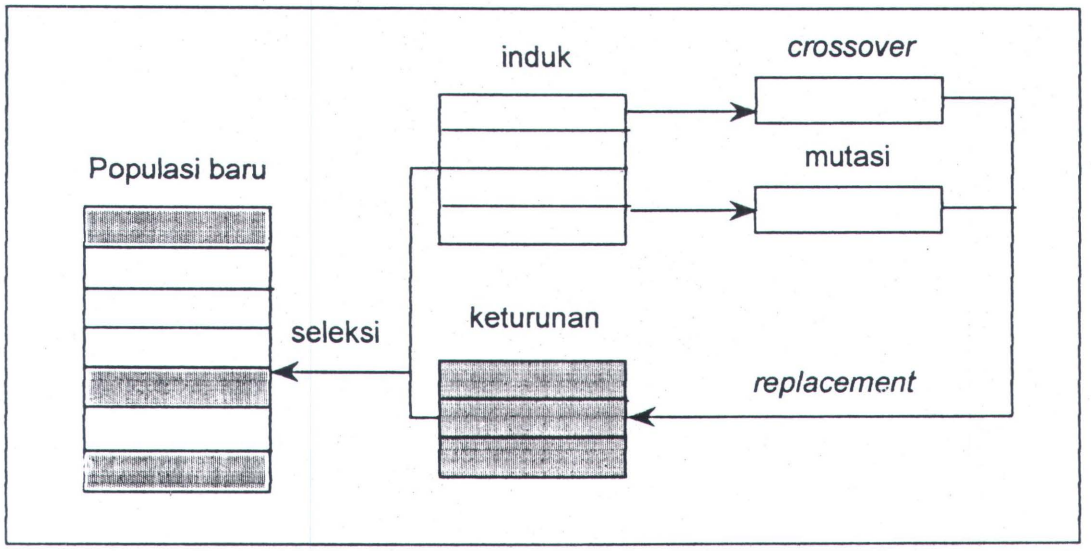
Dengan menganggap bahwa *pop_size* menyatakan ukuran dari populasi dan *off_size* merupakan banyaknya keturunan yang dihasilkan pada suatu generasi.

Daerah sampling tetap mempunyai ukuran sebesar *pop_size* yang terdiri dari semua keturunan dan hanya sebagian induk. Holland menyatakan bahwa induk digantikan keturunannya segera setelah kelahiran keturunan. Pendekatan ini disebut pergantian generasi (*generational replacement*). Karena operasi genetika bersifat tidak pandang bulu, keturunan yang jelek mungkin menggantikan induk yang baik. Pada Gambar 2.7 diberikan gambaran mengenai operasi seleksi pada daerah sampling tetap.



Gambar 2.7 **Penampilan Operasi Seleksi pada Daerah Sampling Tetap**

Daerah Sampling Diperbesar. Dalam proses seleksi berdasarkan daerah sampling yang diperluas, populasi terbentuk dari semua induk dan semua keturunan. Gambar 2.8 memberikan ilustrasi mengenai operasi seleksi pada daerah sampling diperbesar.



Gambar 2.8 **Penampilan Operasi Seleksi pada Daerah Sampling Diperbesar**

Mekanisme Sampling menjawab permasalahan pemilihan kromosom dari daerah sampling. Ada 3 pendekatan umum yang digunakan:

- Sampling Stokastik
- Sampling Deterministik
- Sampling Campuran

Pada pendekatan **Stokastik**, tiap kromosom dipilih berdasarkan kemungkinannya untuk mempertahankan hidup. Salah satu metode yang paling dikenal dari Sampling Stokastik adalah seleksi berdasarkan *roullete wheel*. Ide dasarnya adalah menentukan probabilitas seleksi untuk setiap kromosom berdasarkan nilai fungsinya. Untuk tiap kromosom k dengan nilai fungsi f_k probabilitas seleksi P_k adalah:

$$P_k = \frac{f_k}{\sum_{j=1}^{Pop\_Size} f_j} \qquad (2.1)$$

Sampling Deterministik. Pendekatan ini memilih kromosom terbaik dari daerah sampling. Metode seleksi *elitism* adalah salah satu contoh dari pendekatan sampling deterministik ini. Dalam seleksi elitism, kromosom terbaik diwariskan pada generasi berikutnya.

Sampling Campuran menggabungkan pendekatan Sampling Stokastik dengan Sampling Deterministik. Salah satu contohnya adalah seleksi turnamen. Pada seleksi turnamen, sekelompok kromosom dipilih secara random dan kromosom terbaik dipilih untuk reproduksi.

Probabilitas Seleksi. Pada pendekatan Sampling Stokastik, probabilitas seleksi dari kromosom dihitung berdasarkan nilai fitness-nya. Pendekatan ini memungkinkan beberapa koromosom yang super mendominasi proses seleksi. Untuk mengatasi hal ini dapat dilakukan mekanisme penyesuaian.

Secara umum, nilai fitness setelah disesuaikan f_k' dari nilai fungsi yang sebenarnya f_k untuk kromosom k adalah:

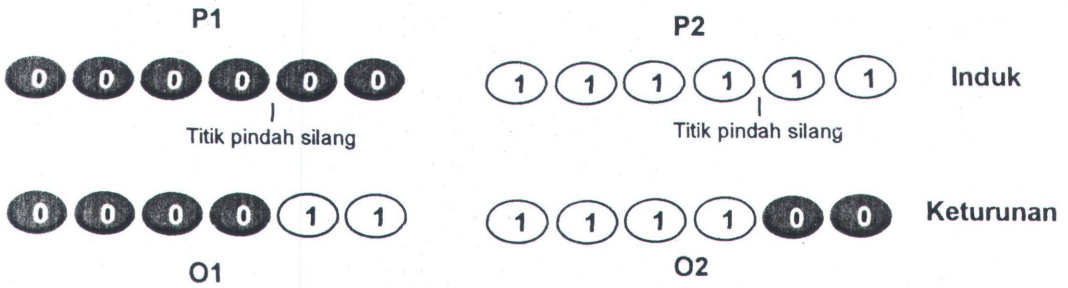
$$f_k' = g\,(f_k)$$

Dimana pada umumnya fungsi *g(.)* bergantung dari permasalahan yang dihadapi.

2.4.2.3 Pindah Silang (*Crossover*)

Pindah silang adalah operator genetik yang utama. Operator ini bekerja dengan mengambil 2 individu dan memotong string kromosom mereka pada posisi yang terpilih secara random, untuk memproduksi dua segment *head* dan dua segment *tail*. Sebagai contoh adalah jika kita mengambil induk yang dipresentasikan dengan 5 dimensi vektor (a_1,b_1,c_1,d_1,e_1) dan (a_2,b_2,c_2,d_2,e_2) kemudian dilakukan *crossing* pada posisi ketiga kromosom-kromosomnya sehingga didapat keturunan (a_1,b_1,c_2,d_2,e_2) dan (a_2,b_2,c_1,d_1,e_1).

Gambar 2.9 merupakan ilustrasi dari operasi pindah silang.



Gambar 2.9 [12]

Ilustrasi Operasi Pindah Silang dalam Algoritma Genetika

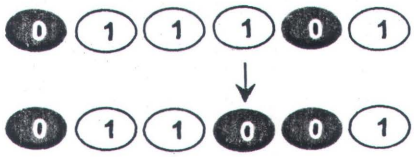
Pindah silang menghasilkan titik baru dalam ruang pencarian yang siap untuk diuji. Operasi ini tidak selalu dilakukan pada semua individu yang ada. Individu dipilih secara random untuk dilakukan crossing dengan P_c antara 0,6 s/d 0,95. Jika pindah silang tidak dilakukan, maka nilai dari induk akan diturunkan kepada keturunan.

2.3.6.4 Mutasi (*Mutation*)

Operator mutasi digunakan untuk melakukan modifikasi satu atau lebih nilai gen dalam individu yang sama. Mutasi memastikan bahwa probabilitas untuk pencarian pada daerah tertentu dalam persoalan tidak akan pernah nol dan mencegah kehilangan total materi genetik setelah pemilihan dan penghapusan. Mutasi ini bukanlah operator genetik yang utama, yang dilakukan secara random pada gen dengan kemungkinan yang kecil (P_m sekitar 0,001). Ilustrasi mengenai cara kerja operator ini digambarkan pada Gambar 2.10.

---

[12] Jose Riberio Filho, Cesare Alippi, Philip Treleaven. *Genetic Algorithm Programming Environments.* University College London, White Paper to appear in the IEEE COMPUTER Journal.



Gambar 2.10 Ilustrasi Operasi Mutasi dalam Algoritma Genetika

2.5 PERBANDINGAN ALGORITMA GENETIKA DENGAN METODE KONVENSIONAL

Seperti yang dipaparkan sebelumnya, Algoritma Genetika merupakan salah satu dari teknik pencarian, yang merupakan cabang dari teknik optimasi algoritma evolusi. Dalam Gambar 2.11 digambarkan mengenai sebagian teknik pencarian yang dikelompokkan menjadi 3 bagian besar : Berbasis *Kalkulus*, *Enumerative* dan pencarian dengan Panduan Bilangan Acak.

Dalam mencari solusi terhadap permasalahan optimasi, ada dua hal yang perlu diperhatikan, yaitu eksploitasi terhadap solusi terbaik dan eksplorasi wilayah pencarian.

- Michalewicz telah melakukan penelitian untuk membandingkan metode pencarian *Hill Climbing*, pencarian dipandu bilangan acak, dan Algoritma Genetika, hasilnya adalah sebagai berikut[13]:
- Teknik Hill Climbing adalah pencarian yang mengeksploitasi solusi terbaik, namun mengabaikan eksplorasi daerah pencarian.

---

[13]Zbigniew Michalewicz. *Genetic Algorithms + Data Structure = Evolution Programs*. Third Revised and Extended Edition, Springer-Verlag, New York, 1996, hal. 15.

- Pencarian yang dipandu bilangan acak adalah pencarian yang mengeksplorasi wilayah pencarian, namun mengabaikan eksploitasi solusi terbaik.
- Algoritma Genetika, pencarian solusi yang optimal menggunakan eksplorasi wilayah pencarian dan eksploitasi solusi terbaik.

Gambar 2.11[14] **Diagram Pengelompokan dalam Teknik Pencarian**

Ada empat perbedaan mendasar yang membedakan Algoritma Genetika dengan teknik optimasi yang konvensional[15]:

1. Mengeksploitasi hasil melalui pengkodean.

   Algoritma Genetika mengeksploitasi hasil atau mengkontrol representasi variabel dengan pengkodean berupa sebuah string, yang mempunyai kinerja tinggi (panjang tetap dan menggunakan allele tertentu). Metode-metode

[14] Op. cit.
[15] Sophia Chan. *Genetic Algorithms, Genetic Algorithm vs Traditional Methods.* White Paper, 1996.

lainnya selalu menggunakan fungsi dan mengkontrol variabel-variabel secara langsung.

2. Dalam melakukan pencarian tidak berangkat dari saru titik awal, melainkan sekumpulan titik yang disebut populasi.

   Pada metode optimasi lain, pencarian biasanya dimulai dari sebuah titik untuk mencari titik berikutnya dengan aturan transisi. Metode ini tidak baik, karena akan mudah menuju puncak yang salah dalam domain *multimodal* (dengan banyak puncak). Algoritma Genetika mencari dari suatu basis data titik (populasi string). Dengan asumsi bahwa sebuah populasi mengadaptasi titik-titik sampel secara baik, sehingga tercapainya nilai optimum pada puncak yang salah, kemungkinannya dapat diperkecil.

3. Mencari dengan memanfaatkan informasi yang menentukan dan tidak memanfaatkan penurunan atau pengetahuan tambahan lain. Algoritma ini dapat dikatakan sebagai sebuah *blind search.*

   Algoritma Genetika secara umum mencapai tujuannya tanpa mempedulikan informasi. Metode lain berjalan dengan sangat bergantung pada beberapa informasi. Jika pada suatu masalah dimana informasi yang dibutuhkan tidak ada atau sulit ditemukan, maka harus diturunkan ke teknik yang lain. Algoritma Genetika bekerja secara umum dengan menggunakan informasi yang ada pada persoalan tersebut. Proses Algoritma Genetika adalah menggunakan nilai yang menentukan dengan pengkodean (string dari tiap individu) yang telah disertai informasinya terhadap urutan struktur-struktur untuk keberlangsungan kemampuan lingkungan masalah yang ada. Dengan menggunakan informasi yang ada, Algoritma Genetika dapat digunakan pada berbagai persoalan.

4. Mencari dengan operator beraturan stokastik, bukan deterministik.

   Di antara operator-operator acak dari Algoritma Genetika dan metode lain, maka Algoritma Genetika bekerja pada bilangan acak secara sederhana. Algoritma Genetika menggunakan bilangan acak untuk mengarahkan pencarian suatu wilayah dalam ruang pencarian dengan *eksploitasi* tinggi (menunjukkan peningkatan).

BAB III
TRAVELING SALESMAN PROBLEM

BAB III

TRAVELING SALESMAN PROBLEM

Pada Bab III ini masih membahas mengenai konsep dasar, yaitu mengenai permasalahan yang diselesaikan pada Tugas Akhir ini, yaitu *Traveling Salesman Problem* (TSP). Pembahasan meliputi TSP itu sendiri yang diteruskan dengan penyelesaian TSP dengan Algoritma Genetika.

3.1 *TRAVELING SALESMAN PROBLEM*

Sebelum kita masuk pada pembahasan penyelesaian TSP dengan Algoritma Genetika, kita harus mengetahui dahulu mengenai TSP ini.

3.1.1 Sejarah[16]

Traveling Salesman Problem adalah persoalan optimasi yang relatif lama, didokumentasikan pertama kali pada awal 1759 oleh Euler, yang mempunyai ketertarikan untuk menyelesaikan problem perjalanan pion perwira dalam permainan catur. Yang mana penyelesaian yang benar menghasilkan sebuah pion perwira dalam permainan catur itu mendatangi hanya sekali dari masing-masing 64 kotak kecil yang ada dalam papan catur.

Istilah *traveling salesman* pertama kali digunakan dalam sebuah buku yang dikeluarkan negara Jerman *The Traveling Salesman, How and*

[16] Zbigniew Michalewicz. *Genetic Algorithms + Data Structure = Evolution Programs.* Third Revised and Extended Edition, Springer-Verlag, New York, 1996, hal. 209.

What He Should Do to Get Commissions and Be Successful in His Bussines pada 1932, yang ditulis oleh seorang veteran *traveling salesman*.

TSP mulai dikenalkan oleh RAND Corporation pada 1948. Selanjutnya dengan gebrakan perusahaan ini, maka TSP dibuat menjadi permasalahan yang terkenal dan populer.

3.1.2 Definisi

Persoalan TSP melibatkan seorang *traveling salesman* (penjaja dagangan) yang harus melakukan kunjungan pada setiap kota yang menjadi bagiannya untuk menjajakan produknya. Rangkaian kota-kota yang dia kunjungi dinamakan lintasan, dimana dalam lintasan tersebut terdapat batasan yaitu tidak boleh ada lebih dari satu kota yang sama. Dengan kata lain, dalam mengunjungi kota-kota, penjaja tidak boleh singgah pada suatu kota lebih dari satu kali. Dan pada akhir perjalanan dia harus kembali ke kota tempat dia memulai perjalanan. Kita bisa ambil contoh misalnya jika terdapat empat kota, yaitu D, E, F, dan G. Lintasan yang ditempuhnya adalah dari kota E ke kota D ke kota G kemudian ke kota F. Setelah sampai di kota F, maka dia harus kembali ke kota E. Penyelesaian dari persoalan ini adalah nilai optimum dari rute yang paling murah, yaitu perjalanan dengan jarak terpendek atau yang mempunyai total harga minimum.

Dari definisi TSP ini, dapat disimpulkan bahwa terdapat dua batasan dalam persoalan ini. Batasan pertama adalah bahwa setiap kota tidak boleh dikunjungi lebih dari satu kali dan pada akhir perjalanan harus kembali ke kota asal, dengan kata lain lintasan berbentuk siklik. Batasan yang kedua bahwa lintasan yang ditempuh adalah lintasan yang paling minimum atau terpendek. Data yang

diketahui dalam permasalahan ini adalah jumlah kota yang harus dikunjungi beserta jarak antar kotanya.

Penyelesaian persoalan ini dengan pendekatan langsung adalah dengan menghitung semua kemungkinan rute yang ada, kemudian dipilih satu rute yang terpendek. Jika ada n kota yang harus dikunjungi maka ada n!/(2n) rute yang harus diselidiki. Dengan cara ini, jumlah waktu komputasi yang diperlukan meningkat seiring dengan bertambahnya ukuran dari persoalan, yaitu jumlah kota. Sebagai contoh, untuk 15 kota akan didapat kurang lebih 4,4 x 10^{10} rute, yang mencerminkan banyaknya rute yang harus diselidiki untuk mendapatkan rute yang optimal sehingga sulit diselesaikan dengan bantuan komputer secara efisien.

Dengan kondisi ini TSP merupakan sebuah permasalahan yang bersifat NPC (*Nondeterministic Polynomial-time Complete*), yang tidak dapat diselesaikan dengan sebuah algoritma linier standar jika kompleksitas problem besar. TSP akan bisa diselesaikan dengan program komputer berkecepatan tinggi jika jumlah node banyak, katakanlah 50. Dengan graph yang besar, maka sebuah algoritma heuristik dibutuhkan.

3.2 PENYELESAIAN DENGAN ALGORITMA GENETIKA

Penerapan Algoritma Genetika untuk menyelesaikan Traveling Salesman Problem pada prinsipnya tidak jauh berbeda dengan struktur Algoritma Genetika secara umum (lihat Bab II, Gambar 2.2). Namun ada beberapa hal yang perlu diperhatikan, yang menunjukkan adanya perbedaan antara Algoritma Genetika konvensional dengan Algoritma Genetika pada penyelesaian TSP (sebagai salah satu permasalahan kombinatorial). Beberapa hal ini perlu diperhatikan untuk

mendapatkan suatu perancangan penyelesaian permasalahan yang sesuai dengan tujuan yang diharapkan. Hal-hal tersebut adalah sebagai berikut:

1. Pengkodekan sebuah lintasan yang sesuai, sehingga penyelesaian mengarah pada hasil yang tepat.
2. Perancangan operator genetik yang dapat diaplikasikan untuk menjaga konsistensi *building blocks* dan mencegah ketidaktepatan
3. Pencegahan konvergensi dini

Langkah pertama yang dilakukan pada peggunaan Algoritma Genetika dalam untuk tujuan penelitian ini adalah memilih pengkodean dengan representasi yang paling sesuai. Kemudian menentukan populasi awal yang akan digunakan dalam mekanisme algoritma. Selanjutnya adalah menentukan fungsi-fungsi yang digunakan oleh algoritma selama mekanisme berlangsung, yaitu fungsi evaluasi dan fungsi seleksi. Dan juga menentukan prosedur operator genetik yang sesuai bagi permasalahan yang diselesaikan. Langkah-langkah ini diperjelas dalam pembahasan subbab-subbab berikutnya.

3.2.1 Pengkodean [17]

Pengkodean sebagai langkah awal dalam penggunaan Algoritma Genetika, dalam penyelesaian TSP ini juga harus dilakukan dahulu. Representasi bilangan biner tidak sesuai (tidak legal) jika diterapkan pada TSP dan permasalahan-permasalahan kombinatorial lainnya. Selama beberapa dekade terakhir ini telah ditemukan beberapa skema representasi yang sesuai untuk permasalahan

[17] Mitsuo Gen, Runwei Cheng. *Genetic Algorithms and Engineering Design.* John Wiley & Sons, Inc., New York, 1997, hal. 118.

kombinatorial. Diantaranya adalah representasi permutasi dan representasi kunci-kunci random.

Representasi Permutasi, representasi ini merupakan representasi yang paling alami untuk lintasan dalam TSP, di mana kota-kota merupakan bagian yang harus dilalui oleh seorang *traveling salesman*. Wilayah pencarian untuk representasi ini adalah himpunan permutasi dari kota-kota berupa simbol abjad. Sebagai contoh, sebuah lintasan dari 9 kota dalam TSP

3 – 2 – 5 – 4 – 7 – 1 – 6 – 9 – 8

Dari representasi di atas dapat diuraikan bahwa lintasan berangkat dari kota 3 dilanjutkan ke kota 2, seterusnya kota 5,4,7,1,6,9, dan diakhiri pada kota 8 yang untuk selanjutnya kembali pada kota 3. Representasi ini sering juga disebut representasi *path* atau representasi *order*.

Representasi Kunci-Kunci Random, representasi ini mengkodekan sebuah *genotype* dengan bilangan-bilangan acak dari (0, 1). Nilai ini digunakan sebagai kunci-kunci untuk mengkodekan penyelesaian. Sebagai contoh, sebuah kromosom untuk 9 kota bisa direpresentasikan

[0.23 0.82 0.45 0.74 0.87 0.11 0.56 0.69 0.78]

di mana posisi i dalam list menunjukkan kota i. Nilai acak dalam posisi i menentukan urutan didatanginya kota i dalam lintasan TSP. Dengan kunci-kunci random di atas, kita dapat menentukan bahwa nilai 0.11 adalah yang paling kecil, sehingga kota ke-6 menempati urutan pertama, 0.23 adalah nilai terkecil kedua,

sehingga kota ke-1 menempati urutan kedua dst. Sehingga dengan demikian, dari kunci-kunci random di atas kita dapat menentukan lintasan:

6 – 1 – 3 – 7 – 8 – 4 – 9 – 2 – 5

Kunci-kunci random dapat memperkecil keturunan yang tidak mungkin dengan merepresentasikannya dalam cara yang sederhana.

3.2.2 Populasi Awal

Populasi awal sangat berpengaruh terhadap kinerja Algoritma Genetika. Apabila dalam populasi awal terdapat kromosom-kromosom yang sudah baik, Algoritma Genetika akan lebih cepat menemukan solusinya. Namun adanya kromosom ini menyebabkan dominasi kromosom super, yang dapat menimbulkan pencarian terjebak pada *local optima*.

Cara inisialisasi awal untuk menghasilkan seluruh kromosom untuk permasalahan kombinatorial ini adalah pembangkitan dengan cara acak.

3.2.2.1 Populasi Awal Acak

Salah satu cara untuk menghasilkan populasi awal acak adalah dengan menggunakan permutasi Josephus. Misalkan ada kota dari 1 sampai 9. Permutasi dari lintasan dapat dilakukan dengan menentukan titik awal dan selang. Misalnya titik awal adalah 6 dan selang adalah 5. Maka lintasan berangkat dari kota 6, selang 5 dari kota 6 adalah kota 2 (dengan asumsi kota 1 sampai 9 membentuk *circular list*). Kota 2 dihapus dari list. Selang 5 kemudian adalah kota 7. Proses ini

diulang hingga ada satu lintasan dalam list. Hasil dari permutasi ini adalah 2 – 7 – 3 – 8 – 4 – 9 – 5 – 1 – 6.

3.2.3 Fungsi Penilaian

Untuk mengevaluasi kromosom-kromosom, maka digunakan fungsi penilaian sebagai berikut:

$$eval(v_k) = \sum_{i=1}^{city\_size} dist(c_i - c_{i+1}) + dist(c_{city\_size} - c_1) \qquad (3.1)$$

dimana city_size adalah jumlah kota dalam lintasan/kromosom dan $dist(c_i - c_{i+1})$ adalah jarak antara kota c_i dengan kota c_{i+1} yang ada dalam lintasan.

3.2.4 Seleksi

Seleksi yang digunakan adalah sampling stokastik dengan wilayah yang tetap.

3.2.4.1 Piringan *Roullete*

Sampling stokastik diimplementasikan dengan piringan *roullete*. Berikut ini adalah langkah-langkah untuk proses seleksi:

Prosedur pra-seleksi

Langkah 1 Hitung nilai *fitness* bagi tiap kromosom

$$eval\ (v_i) = f(x), \qquad i = 1,2,...,pop\_size$$

Langkah 2 Hitung total nilai *fitness*

$$F = \sum_{i=1}^{pop\_size} eval(v_i) \qquad (3.2)$$

Langkah 3 Hitung probabilitas seleksi P_i untuk tiap kromosom

$$P_i = \frac{eval(v_i)}{F} \qquad i = 1,2,\ldots,pop\_size \qquad (3.3)$$

Langkah 4 Hitung probabilitas komulatif q_i untuk tiap kromosom

$$q_i = \sum_{j=1}^{i} p_j \qquad i = 1,2,\ldots,pop\_size \qquad (3.4)$$

Prosedur seleksi

Langkah 1 Bangkitkan bilangan acak r antara 0 dan 1

Langkah 2 Jika $r \leq q_1$ maka pilih kromosom pertama (v_1). Jika tidak, pilih kromosom v_i ($2 \leq i \leq pop\_size$) sedemikian rupa sehingga $q_{i-1} < r \leq q_i$.

3.2.5 Mekanisme Penyesuaian

Untuk menghindarkan dominasi individu tertentu pada proses Algoritma Genetika, maka digunakan beberapa mekanisme penyesuaian, diantaranya seperti yang disebutkan di bawah ini.

Windowing. Pada mekanisme ini nilai *fitness* setiap kromosom dikurangi dengan nilai *fitness* yang terkecil. Mekanisme ini memperbesar probabilitas seleksi dari kromosom yang paling kuat, akan tetapi menghilangkan kromosom yang paling lemah.

Eksponensial. Nilai *fitness* setiap kromosom ditambah 1, kemudian dikuadratkan. Mekanisme ini memperbesar probabilitas seleksi dari kromosom-kromosom yang relatif lemah.

Normalisasi Linier (*Linear Normalization*), Bertujuan untuk mempertinggi kemampuan kromosom terbaik untuk reproduksi di saat kromosom yang lemah justru memungkinkan untuk memproduksi keturunan. Sebagai contoh adalah dengan *base value* 20, *decrement* 8 dan nilai minimum 1 Normalisasi Linier akan menunjukkan nilai fitness-nya seperti dalam Tabel 3.1.

Tabel 3.1 Tabel Contoh Mekanisme Penyesuaian

Kromosom	Fitness Awal	Windowing	Eksponensial	Normalisasi Linier
1	0.255 (10%)	0.000 (0%)	1.575 (13%)	1.00 (3%)
2	0.733 (29%)	0.518 (37%)	3.144 (26%)	12.00 (31%)
3	0.405 (15%)	0.150 (11%)	1.974 (16%)	4.00 (10%)
4	0.928 (35%)	0.673 (48%)	3.717 (31%)	20.00 (53%)
5	0.318 (11%)	0.063 (4%)	1.737 (14%)	1.00 (3%)

Untuk mekanisme windowing dan eksponensial, mekanisme ini berlaku untuk pencarian nilai maksimum. Jika mekanisme ini diterapkan pada TSP dimana yang dicari adalah nilai minimum, yang dilakukan adalah memaksimumkan nilai eval yang didapatkan dalam masing-masing kromosom. Sebagai contoh jika didapatkan eval = f_x maka maksimum dari nilai tersebut adalah $1/f_x$.

3.2.6 Operator-Operator Algoritma Genetika

Perbedaan yang nampak sekali Algoritma Genetika konvensional dengan Algoritma Genetika yang digunakan dalam permasalahan-permasalahan kombinatorial adalah pada metode tukar silang dan mutasi.

3.2.6.1 Pindah Silang

Jika dalam bab sebelumnya telah diperlihatkan teknik pindah silang dimana sepasang kromosom diproduksi dengan mengkopi komplemen induk pada keturunan. Kita bisa melihat untuk *string* berupa angka biner, teknik pindah silang ini bekerja dengan baik. Akan tetapi apa yang terjadi jika pindah silang satu-poin atau dua-poin kita terapkan pada list yang mempresentasikan tujuan perjalanan antar kota, hal tersebut ditunjukkan pada Tabel 3.1.

Tabel 3.1 **Tabel hasil Pindah Silang Konvensional pada TSP**

	Induk	**Keturunan**
#1	123|45678	123|21345
#2	768|21345	768|45678

	Induk	**Keturunan**
#1	12|345|678	12|821|678
#2	76|821|345	76|345|345

Sesuatu yang terjadi jika pindah silang konvensional diterapkan untuk permutasi TSP adalah tidak dapat diterima hasil-hasil yang didaptkan oleh permasalahan ini. Secara lebih jelas pada lintasan yang ditunjukkan oleh keturunan kedua *traveling salesman* akan melakukan pengulangan pada kota 3,4,5

dan kehilangan kota 1,2,8. Hal ini menjadikan daerah solusi sebagai daerah *infeasible*.

Sejak pertengahan 80'an, beberapa metode operator pindah silang diciptakan untuk representasi permutasi, seperti *partial-mapped crossover*, *order crossover*, *cycle crossover*, *position-based crossover*, *order-based crossover*, *heuristic crossover*, dll. Beberapa metode pindah silang tersebut dijelaskan seperti di bawah ini.

Partial-Mapped Crossover (PMX). PMX diciptakan oleh Goldberg dan Lingle. PMX merupakan rumusan modifikasi dari pindah silang dua-poin. Hal yang penting dari PMX adalah pindah silang 2-poin ditambah dengan beberapa prosedur tambahan. PMX mempunyai langkah kerja sebagai berikut:

Prosedur PMX

Langkah 1 : Tentukan dua posisi pada kromosom dengan aturan acak. Substring yang berada dalam dua posisi ini dinamakan daerah pemetaan.

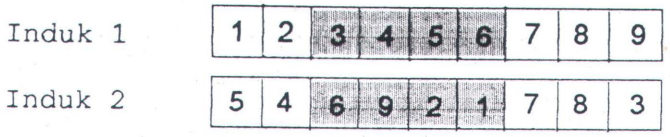
Langkah 2 : Tukar dua substring antar induk untuk menghasilkan *proto-child*.

Langkah 3 : Tentukan hubungan pemetaan di antara dua daerah pemetaan.

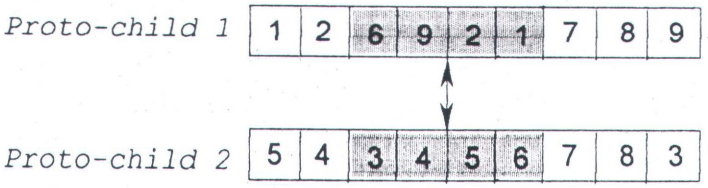
Langkah 4 : Tentukan kromosom keturunan mengacu pada hubungan pemetaan.

Prosedur ini dapat dilihat ilustrasinya pada Gambar 3.1.

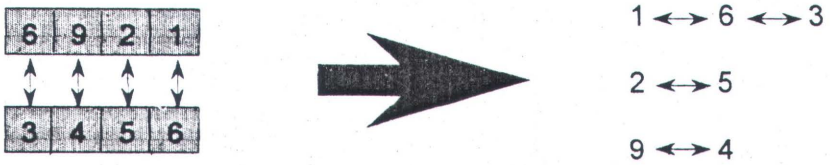
1. Pilih posisi untuk menentukan substring secara acak



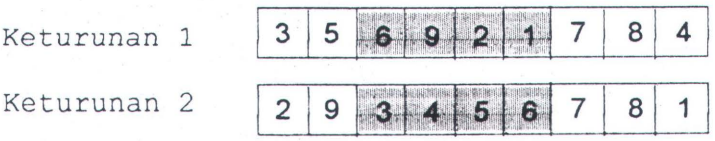
2. Tukar substring di antara induk



3. Menentukan hubungan mapping



4. Menentukan kromosom keturunan mengacu pada hubungan mapping



Gambar 3.1[18] **Ilustrasi dari PMX operator**

Order Crossover (OX). OX diciptakan oleh Davis. Metode ini merupakan variasi dari PMX dengan prosedur tambahan. OX bekerja sbb:

Prosedur OX

Langkah 1 : Pilih substring dari sebuah induk secara acak.

[18] Ibid., hal. 121.

Langkah 2 : Bangkitkan sebuah *proto-child* dengan mengkosongkan tempat substring induk 2 pada induk 1.

Langkah 3 : SHR allele dari substring pada tempat yang bersesuaian.

Langkah 4 : Tukar substring antara 2 induk.

Gambar 3.2 adalah ilustrasi dari OX.

Cycle Crossover (CX). CX diciptakan oleh Oliver, Smith dan Holland. Metode ini mengkopi kota-kota dari satu induk dan memilih kota-kota yang lain dari induk yang lain, dengan mengingat dan pola cycle. Cara kerja CX adalah sbb:

Prosedur CX

Langkah 1 : Temukan cycle yang didefinisikan dari relasi posisi kota-kota antara induk

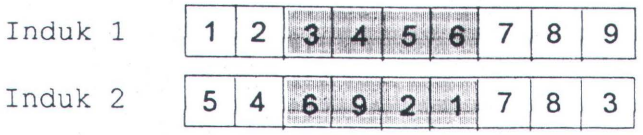
Langkah 2 : Salin kota-kota dalam *cycle* pada *proto-child* dengan relasi posisi dari sebuah induk

Langkah 3 : Tentukan kota-kota diingat yang berasal dari induk lain

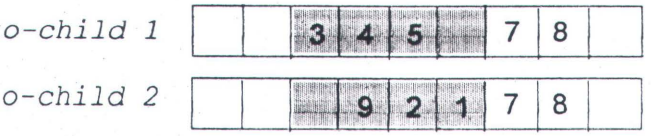
Langkah 4 : Isi keturunan dengan kota-kota yang diingat tadi

Gambar 3.3 adalah ilustrasi dari CX .

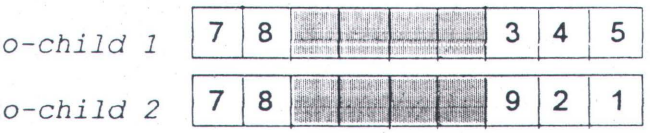
1. Memilih substring dari induk dengan cara acak



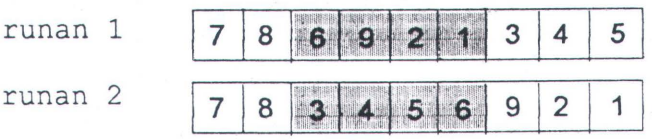
2. Produksi *proto-child* dengan mengosongkan tempat substring induk 2 pada induk 1



3. SHR substring pada tempat yang bersesuaian



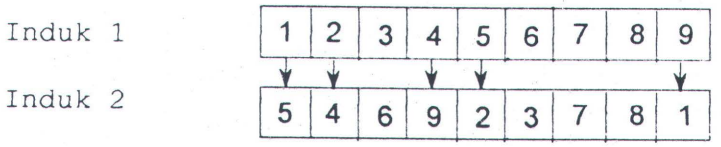
4. Tukar posisi substring



Gambar 3.2[19] Ilustrasi OX operator

[19] Scott Robert Ladd. *Genetic Algorithms in C++*. M&T Books, New York, 1996, hal. 143.

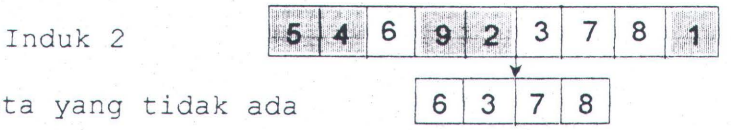
1. Tentukan pola *cycle* (asumsi : pola dimulai dari posisi 1)



Pola *cycle* $1 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 9 \rightarrow 1$

2. Kopi kota-kota dalam *cycle* pada *proto-child*

3. Tentukan kota-kota yang diingat dari induk yang lain



4 Mengisikan kota yang diingat dalam keturunan

Dengan cara yang sama, diperoleh keturunan 2

Gambar 3.3.[20] Ilustrasi CX operator

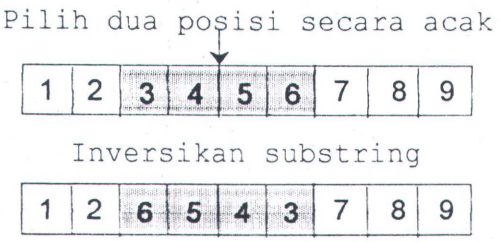
3.2.6.2 MUTASI[21]

Hal yang relatif mudah ketika kita akan menciptakan operator mutasi untuk representasi permutasi. Selama dekade terakhir, beberapa operator mutasi telah

[20] Op. cit., hal. 124.
[21] Ibid., hal. 125.

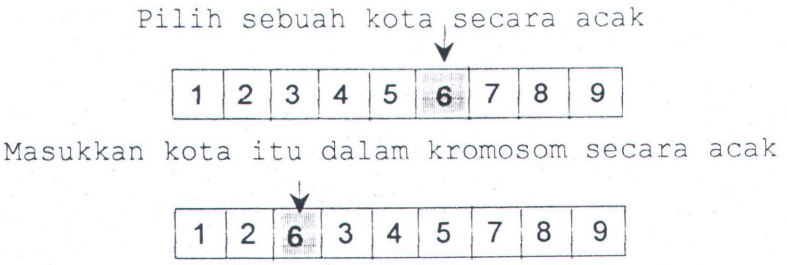
diciptakan untuk representasi permutasi, seperti metode *inversion, insertion, displacement, dan reciprocal exchange mutation.*

Inversion Mutation. *Inversion mutation* memilih dua posisi dalam sebuah kromosom dengan cara acak dan kemudian menginversikan substring di antara dua posisi tersebut. Gambar 3.4. adalah ilutrasi dari *Inversion Mutation*.



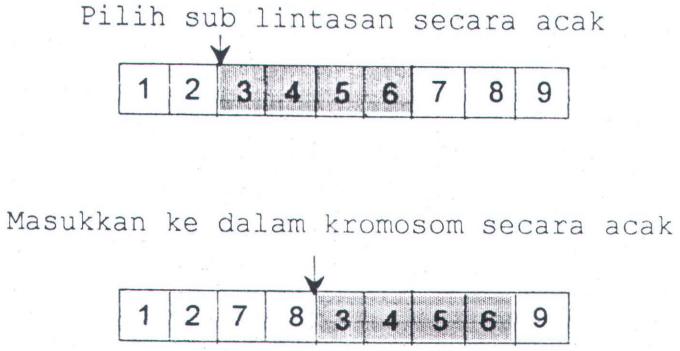
Gambar 3.4 Ilustrasi ***Inversion Mutation***

Insertion Mutation. *Insertion Mutation* memilih sebuah kota dengan cara acak dan memasukkan ke dalam kromosom dengan cara acak pula, seperti diilustrasikan dalam Gambar 3.5.



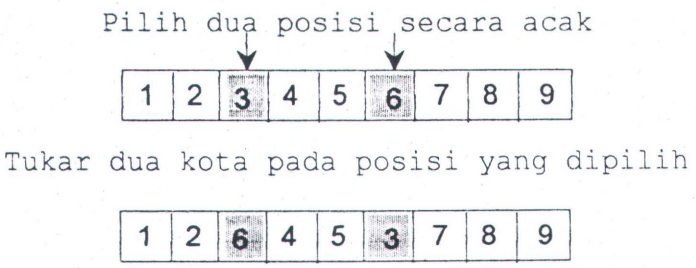
Gambar 3.5 Ilustrasi ***Insertion Mutation***

Displacement Mutation. *Displacement Mutation* memilih sebuah sub lintasan dengan cara acak kemudian memasukkan ke dalam kromosom dengan cara acak, seperti diilustrasikan pada Gambar 3.6.



Gambar 3.6 Ilustrasi ***Displacement Mutation***

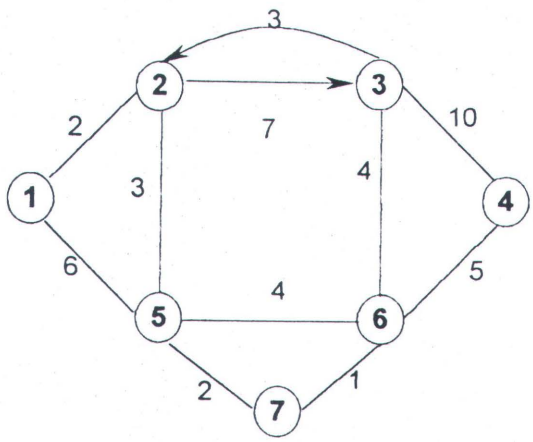
Reciprocal Exchange Mutation. REM memilih dua posisi secara acak, kemudian menukar dua kota dalam posisi tersebut seperti diilustrasikan pada Gambar 3.7.



Gambar 3.7 Ilustrasi ***Reciprocal Exchange Mutation***

3.3 CONTOH SEDERHANA PENYELESAIAN TSP

Pada bagian ini dijelaskan mengenai kinerja Algoritma Genetika dalam menyelesaikan sebuah contoh TSP. Selain itu diuraikan bagaimana metode lain, dalam hal ini Branch and Bound menyelesaikan contoh tersebut. Contoh yang diambil adalah seperti pada Gambar 3.8.



Gambar 3.8 Peta Sebuah Contoh TSP

3.3.1 Penyelesaian dengan Algoritma Genetika

Dari gambar peta di atas, Algoritma Genetika menyelesaikan dengan langkah-langkah sebagai berikut:

1. Membangun populasi. Untuk membangun populasi ini kromosom dikodekan dengan representasi permutasi. Untuk peta di atas tiap kromosom dikodekan dengan 7 buah gen (karena jumlah kota dalam tiap-tiap lintasan adalah 7), dimana allele yang dimasukkan dalam tiap kromosom adalah simbol tiap kota berupa abjad. Ilustrasi pengkodean ini dapat dilihat pada Gambar 3.9. Langkah-langkah membangun populasi adalah sebagai berikut:

- Untuk kromosom pertama dibentuk dengan membangun bilangan acak sebanyak 7 kali. Dari bilangan acak ini dibentuk kromosomnya dalam representasi permutasi. Sebagai contoh misalnya didapatkan bilangan acak sebagai berikut:

  [0.34 0.67 0.11 0.29 0.91 0.56 0.77]

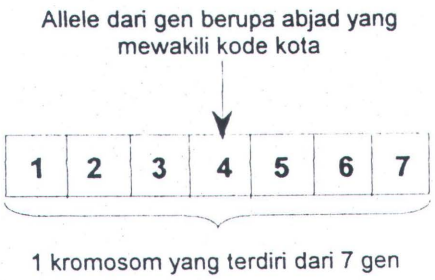
  maka didapatkan kromosom sebagai berikut:

  3 – 4 – 1 – 6 – 2 – 7 – 5.

- Setelah itu dilakukan pengecekan kebenaran lintasan yang diwakili kromosom di atas, pengecekan dilakukan dengan melihat data jarak antar kota. Apabila antara 2 kota tidak ada lintasan yang menghubungkan secara langsung, nilai yang ditunjuk untuk jarak keduanya adalah nilai yang besar untuk tipe data yang digunakan. Dan apabila dalam suatu lintasan terdapat dua kota yang tidak terhubung secara langsung, maka lintasan ini dianggap tidak benar. Lintasan di atas tidak benar karena antara kota 4 dan 1 tidak lintasan yang langsung. Langkah ini terus diulang sampai didapatkan kromosom yang mewakili lintasan yang benar. Misalnya didapatkan v_1 = 3–4–6–7–5–1–2.

- Untuk kromosom kedua dan seterusnya maka dibangun dengan permutasi Josephus (seperti yang sudah dijelaskan dalam subbab 3.2.2), dengan langkah-langkah yang dilakukan pertama kali adalah membangkitkan dua bilangan bulat acak yaitu $1 \leq s \leq 7$ dan $1 \leq j < 7$. Dimana s adalah untuk menentukan titik awal dari lintasan yang baru. Sedangkan j digunakan untuk selang kota. Misalnya dari proses ini dihasilkan bilangan s=2 maka lintasan yang dicari berangkat dari kota 2, kemudian j=3, sehingga selang 3 dari 2 adalah kota 5, selang 3 kemudian adalah kota 1, seterusnya hingga didapat

lintasan v_2= 2–5–1–4–7–3–6. Kemudian kromosom ini dicek apakah mewakili lintasan yang benar atau tidak, jika tidak maka kromosom pertama (sebagai kromosom yang benar) dikopikan sebagai kromosom kedua. Kemudian membangkitkan lagi bilangan acak untuk kromosom berikutnya. Langkah ini dilakukan sebanyak jumlah populasi - 1, dimana jumlah populasi merupakan angka yang ditunjuk oleh input parameter Algoritma Genetika (penjelasan mengenai parameter lihat subbab 2.3.2). Sebagai contoh dari persoalan ini dengan jumlah populasi = 7 didapat populasi dengan kromosom v_2 = 3–4–6–7–5–1–2, v_3 = 3–4–6–7–5–1–2, v_4 = 3–4–6–7–5–1–2, v_5 = 3–4–6–7–5–1–2, v_6 = 3–4–6–7–5–1–2, v_7 = 3–4–6–7–5–1–2, v_8 = 3–4–6–7–5–1–2, v_9 = 3–4–6–7–5–1–2, v_{10} = 3–4–6–7–5–1–2.



Gambar 3.9 Pengkodean Contoh TSP

2. Dari populasi yang telah dibangun kemudian dicari fungsi penilaian tiap kromosom. Dengan formula 3.1 didapatkan hasil evaluasi dari tiap kromosom sebagai berikut:

$$eval(v_1) = dist_{3-4} + dist_{4-6} + dist_{6-7} + dist_{7-5} + dist_{5-1} + dist_{1-2} + dist_{2-3}$$

$$= 10 + 5 + 1 + 2 + 6 + 2 + 7 = 33$$

$$eval(v_2) = dist_{3-4} + dist_{4-6} + dist_{6-7} + dist_{7-5} + dist_{5-1} + dist_{1-2} + dist_{2-3}$$

$$= 10 + 5 + 1 + 2 + 6 + 2 + 7 = 33$$

$$eval(v_3) = dist_{3-4} + dist_{4-6} + dist_{6-7} + dist_{7-5} + dist_{5-1} + dist_{1-2} + dist_{2-3}$$

$$= 10 + 5 + 1 + 2 + 6 + 2 + 7 = 33$$

$$eval(v_4) = dist_{3\text{-}4} + dist_{4\text{-}6} + dist_{6\text{-}7} + dist_{7\text{-}5} + dist_{5\text{-}1} + dist_{1\text{-}2} + dist_{2\text{-}3}$$

$$= 10 + 5 + 1 + 2 + 6 + 2 + 7 = 33$$

$$eval(v_5) = dist_{3\text{-}4} + dist_{4\text{-}6} + dist_{6\text{-}7} + dist_{7\text{-}5} + dist_{5\text{-}1} + dist_{1\text{-}2} + dist_{2\text{-}3}$$

$$= 10 + 5 + 1 + 2 + 6 + 2 + 7 = 33$$

$$eval(v_6) = dist_{3\text{-}4} + dist_{4\text{-}6} + dist_{6\text{-}7} + dist_{7\text{-}5} + dist_{5\text{-}1} + dist_{1\text{-}2} + dist_{2\text{-}3}$$

$$= 10 + 5 + 1 + 2 + 6 + 2 + 7 = 33$$

$$eval(v_7) = dist_{3\text{-}4} + dist_{4\text{-}6} + dist_{6\text{-}7} + dist_{7\text{-}5} + dist_{5\text{-}1} + dist_{1\text{-}2} + dist_{2\text{-}3}$$

$$= 10 + 5 + 1 + 2 + 6 + 2 + 7 = 33$$

$$eval(v_8) = dist_{3\text{-}4} + dist_{4\text{-}6} + dist_{6\text{-}7} + dist_{7\text{-}5} + dist_{5\text{-}1} + dist_{1\text{-}2} + dist_{2\text{-}3}$$

$$= 10 + 5 + 1 + 2 + 6 + 2 + 7 = 33$$

$$eval(v_9) = dist_{3\text{-}4} + dist_{4\text{-}6} + dist_{6\text{-}7} + dist_{7\text{-}5} + dist_{5\text{-}1} + dist_{1\text{-}2} + dist_{2\text{-}3}$$

$$= 10 + 5 + 1 + 2 + 6 + 2 + 7 = 33$$

$$eval(v_{10}) = dist_{3\text{-}4} + dist_{4\text{-}6} + dist_{6\text{-}7} + dist_{7\text{-}5} + dist_{5\text{-}1} + dist_{1\text{-}2} + dist_{2\text{-}3}$$

$$= 10 + 5 + 1 + 2 + 6 + 2 + 7 = 33$$

Selanjutnya fungsi *fitness* mencatat lintasan dengan hasil terkecil dari fungsi evaluasi ini. Dari contoh di atas, maka kromosom v_1 dengan hasil eval = 33 tercatat dalam variabel *fit*.

3. Sebelum melakukan proses seleksi, dilakukan dulu salah satu mekanisme penyesuaian seperti yang telah dijelaskan dalam subbab 3.2.5. Misalnya dari contoh di atas digunakan Normalisasi Linier untuk skalanisasi nilai-nilai fitness, dengan base = 100, decrement = 20, minimal = 1. Dengan formula 3.2 didapatkan total *fitness F* dari populasi awal adalah:

$$F = \sum_{k=1}^{10} eval(v_k) = 330$$

Dari hasil evaluasi yang didapatkan maka dilakukan sorting untuk persiapan Mekanisme Penyesuaian. Kemudian dari sini dilakukan Mekanisme Penyesuaian dan didapatkan hasil sebagai berikut:

Tabel 3.2 **Tabel Hasil Penyesuaian dari Contoh TSP**

Kromosom	Fitness Awal	Normalisasi Linier
1	33 (10%)	100 (32.785%)
2	33 (10%)	80 (26.23%)
3	33 (10%)	60 (19.67%)
4	33 (10%)	40 (13.11%)
5	33 (10%)	20 (6.56%)
6	33 (10%)	1 (0.325%)
7	33 (10%)	1 (0.325%)
8	33 (10%)	1 (0.325%)
9	33 (10%)	1 (0.325%)
10	33 (10%)	1 (0.325%)

Catatan : Nilai di dalam tanda kurung dalam tabel 3.2 adalah probabilitas baru (P_k) yang didapat dengan formula 3.3.

4. Selanjutnya dilakukan langkah seleksi. Dari hasil penyesuaian di atas dan mengacu pada prosedur Roulette Wheel yang telah dijelaskan pada subbab 3.2.4.1 maka dilakukan langkah-langkah untuk prosedur pra seleksi sebagai berikut:
   - Menghitung eval(v_k) untuk masing-masing kromosom. Hasilnya telah diketahui bersamaan dengan langkah 2 di atas
   - Menghitung total *fitness* dari populasi. Hasilnya adalah 305.

- Menghitung probabilitas P_k untuk masing-masing kromosom v_k. Untuk langkah ini, maka P_k yang digunakan adalah hasil yang didapat pada langkah 3 (setelah dilakukan penyesuaian). Dengan penyesuaian Normalisasi Linier, maka didapatkan:

P_1 = 100 (32.785%)	P_6 = 1 (0.325 %)
P_2 = 80 (26.23%)	P_7 = 1 (0.325 %)
P_3 = 60 (19.67%)	P_8 = 1 (0.325 %)
P_4 = 40 (13.11%)	P_9 = 1 (0.325 %)
P_5 = 20 (6.56%)	P_{10} = 1 (0.325 %)

- Menghitung probabilitas komulatif q_k untuk masing-masing kromosom v_k (k = 1...10) sesuai dengan formula 3.4, sehingga didapatkan

q_1 = 0.32785	q_6 = 0.98680
q_2 = 0.59015	q_7 = 0.99005
q_3 = 0.78685	q_8 = 0.99330
q_4 = 0.91795	q_9 = 0.99655
q_5 = 0.98355	q_{10} = 1.000

Kemudian lakukan prosedur seleksi:

- Membangkitkan bilangan acak [0,1] sebanyak 10 buah dan didapatkan

0.753	0.258	0.779	0.442	0.742
0.226	0.720	0.958	0.281	0.373

- Angka pertama r_1 = 0.753 lebih kecil dari q_3 = 0.78685, sehingga kromosom ketiga (v_3) dipilih untuk kromosom baru yang pertama, selanjutnya r_2 = 0.258 lebih kecil juga dari q_1 = 0.32785, sehingga kromosom pertama (v_1) dipilih untuk kromosom kedua. Angka ketiga r_3 =

0.779 lebih besar dari q_3 dan lebih kecil dari q_4 sehingga v_4 terpilih untuk kromosom ketiga. Sedang angka keempat r_3 = 0.442 lebih besar dari q_1 dan lebih kecil dari q_2 sehingga v_2 terpilih untuk kromosom keempat. Atau hasil selengkapnya adalah sbb.:

$v_1' = v_3$ $\quad$ $v_6' = v_1$

$v_2' = v_1$ $\quad$ $v_7' = v_3$

$v_3' = v_3$ $\quad$ $v_8' = v_5$

$v_4' = v_2$ $\quad$ $v_9' = v_1$

$v_5' = v_3$ $\quad$ $v_{10}' = v_2$

5. Langkah berikutnya rekombinasi Genetika dengan menggunakan operator-operator Genetika. Yang pertama dilakukan adalah pindah silang (*crossover*). Operasi ini dilakukan dengan memperhatikan probabilitas pindah silang (P_c), yang merupakan masukan dalam parameter Algoritma Genetika (lihat subbab 2.3.2). Selain itu operasi ini dilakukan dengan melihat juga metode pindah silang yang dipilih. Jika dimisalkan dalam contoh ini P_c = 0.8 dan metode pindah silang yang dipilih adalah metode CX, maka prosesnya adalah sebagai berikut:

   - Menghitung jumlah struktur yang ikut berperan dalam operasi pindah silang. Seperti telah dijelaskan dalam subbab 2.3.2, bahwa jumlah struktur yang melakukan pindah silang adalah:

     $$P_c * Pop\_size = 0.8 * 10 = 8$$

   - Dengan metode CX yang dilakukan pada 8 kromosom yang dipilih dengan cara acak yang unik, misalnya didapatkan angka 1,4,3,5,6,10,2 dan 8.

Untuk pasangan kromosom pertama yang akan dilakukan pindah silang yaitu 1 & 4, kemudian untuk pasangan kromosom kedua yaitu 3 & 5, dan pasangan kromosom ketiga yaitu 6 & 10 dan pasangan keempat adalah kromosom 2 & 8. Karena untuk kasus ini tiap-tiap kromosom yang dihasilkan sampai pada langkah di atas polanya sama, maka dengan operator pindah silang akan tetap didapatkan pola yang sama, yaitu untuk kromosom pertama sampai dengan kromosom kesepuluh didapatkan pola v_i = 3–4–6–7–5–1–2. Dengan hasil konvergensi dini seperti ini, mekanisme Algoritma Genetika mengandalkan operator Mutasi untuk menghindarinya.

6. Kemudian dilakukan operasi mutasi dengan berdasarkan pada probabilitas mutasi (P_m) dan jenis operator yang digunakan (lihat subbab 3.2.6). Misalnya dalam contoh i P_m = 0.05 dan metode mutasi yang dipilih adalah Inversion. Dihitung jumlah allele yang melakukan mutasi, didapatkan:

$$P_m * \textit{Jumlah allele} = 0.05 * 5 * 10 = 2.5 \approx 3$$

Mutasi Inversion dilakukan dengan membangkitkan bilangan acak antara 1 sampai dengan jumlah keseluruhan allele (ukuran populasi * jumlah kota), misalnya didapatkan 23, maka dilakukan mutasi pada allele ke-23. Untuk menghitung allele ke-23 terletak pada kromosom ke berapa, maka bilangan 23 ini dibagi dengan bilangan 7 dengan hasil bilangan yang bulat. Untuk menghitung allele keberapa dari kromosom ini, maka bilangan 23 di-modulasi-kan dengan bilangan 7. Sehingga mutasi dilakukan pada kromosom ketiga pada allele kedua. Kemudian untuk kinerja mutasi Inversion maka dibangkitkan lagi bilangan acak antara 1 sampai 7. Misalnya dari sini didapatkan bilangan 7, maka pada kromosom ketiga pada allele kedua sampai ketujuh dilakukan

mutasi Inversion. Sehingga didapatkan kromosom baru v_3' = 3–2–1–5–7–6–4. Langkah mutasi Inversion ini dilakukan sebanyak 3 kali, hingga didapatkan :

v_7' = 3–4–6–1–5–7–2

v_{10}' = 3–7–6–4–5–1–2

7. Dari langkah keenam ini kemudian v_i dan v_i' dicampur dan dilakukan seleksi terhadap kromosom terbaik sebanyak ukuran populasi, sehingga dari sini kromosom yang mewakili lintasan yang tidak benar akan hilang dengan sendirinya.
8. Dari populasi baru ini iterasi kembali pada langkah ke-2 dan iterasi dilakukan sebanyak jumlah generasi.
9. Hasil optimum didapatkan pada generasi kedua, dengan lintasan = 3–2–1–5–7–6–4–3 dan panjang lintasan = 29.

Seperti yang dipaparkan sebelumnya, bahwa lintasan yang dibentuk oleh TSP ini berbentuk siklik. Sehingga apabila didapatkan lintasan 3–2–1–5–7–6–4–3 seperti pada poin sembilan di atas, maka lintasan tersebut sebenarnya mempunyai panjang lintasan yang sama (panjang lintasan = 29) dengan lintasan yang dimulai dari kota 6, yaitu lintasan 6–4–3–2–1–5–7–6.

3.3.2 Penyelesaian dengan Branch and Bound

Algoritma Branch and Bound yang dipilih di sini adalah Branch and Bound With Underestimate.[22] Algoritma ini dimulai dari suatu kota, kemudian dari kota ini dibuat *tree*(*branch*). Tree yang dibentuk adalah *binary tree* dengan branch berupa

---

[22] Patrick Henry Winston. *Artificial Intelligence.* Second Edition, Addison-Wesley Publishing Company, Masachussetts, 1984, hal 107.

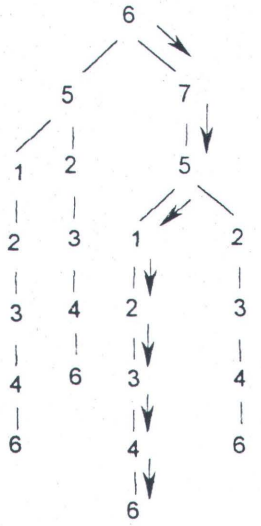
lintasan dua kota yang mempunyai jarak (jarak parsial + jarak underestimate) yang terpendek pertama.

Dari contoh di atas kita menentukan kota *start*, misalnya kota 6, dari kota 6 kemungkinan tree-nya adalah ke kota 5, 7. Kemudian dilihat jarak kota 6 ke kota-kota tersebut dan juga dilihat jarak underestimate-nya (jarak kota-kota tersebut dengan kota 6). Didapatkan :

$$dist_{6-5} + dist_{5-6} = 4 + 4 = 8,$$

$$dist_{6-7} + dist_{7-6} = 1 + 1 = 2,$$

Dari hasil ini kemudian disorting dan dua urutan pertama (6 – 7, 6 – 5) dimasukkan tree, kemudian dibentuk lintasan parsialnya yang dimasukkan sebuah *queue*. Diadapatkan isi queue yaitu [6 – 7, 6 – 5]. Kemudian dari lintasan parsial yang pertama dibentuk kemungkinan tree-nya lagi, didapatkan 6 – 7 – 5. Dihitung lagi jarak parsial + underestimatenya untuk masing-masing lintasan parsial, didapatkan 7. Dua yang terkecil dimasukkan queue (karena kali ini lintasan parsial yang terbentuk hanya satu, maka satu lintasan tersebut yang dimasukkan queue). Kemudian dilakukan pengurutan, sehingga isi queue menjadi [6 – 7 – 5, 6 – 5] dan dilakukan proses yang sama dari lintasan parsial pertama yang ada dalam queue. Terus diiterasi sampai didapatkan hasil dimana isi queue yang pertama telah melintasi semua kota. Sehingga didapatkan nilai optimum adalah 33 dengan lintasan 6 – 7 – 5 – 1 – 2 – 3 – 4 – 6. Tree yang terbentuk dari contoh ini dapat dilihat dalam Gambar 3.10.



Gambar 3.10 Tree yang Dibentuk oleh Metode Branch and Bound dari Contoh TSP

BAB IV
PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK

BAB IV

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK

Dalam bab ini dibahas mengenai perancangan dan pembuatan perangkat lunak, yang dimulai dengan pembahasan mengenai perancangan data, diteruskan dengan perancangan proses, perancangan antarmuka dan implementasi dalam bahasa pemrograman.

4.1 PERANCANGAN DATA

Dalam perancangan suatu perangkat lunak ada dua bagian utama yaitu data dan proses. Pada perancangan perangkat lunak kali ini secara garis besar data dikelompokkan menjadi tiga bagian, yaitu data masukan, data proses dan data keluaran.

4.1.1 Data Masukan

Data masukan yang digunakan perangkat lunak ini berasal dari berkas masukan yang bertipe teks. File ini berisi informasi jumlah kota dari TSP yang akan diselesaikan. Selain itu dalam file ini berisi juga mengenai informasi jarak antar kota. Tata bahasa yang digunakan untuk menyusun deskripsi data masukan adalah seperti yang tertera di bawah ini.

```
[TSP Parameter]
Number of cities = 7

[Distances between cities]
1 2 = 2
2 3 = 7
3 4 = 10
4 6 = 5
3 6 = 4
6 5 = 4
6 7 = 1
5 7 = 2
1 5 = 6
2 5 = 3
2 1 = 2
3 2 = 7
4 3 = 10
6 4 = 5
6 3 = 4
5 6 = 4
7 6 = 1
7 5 = 2
5 1 = 6
5 2 = 3
3 2 = 3
end
```

Keterangan untuk masing-masing data adalah sebagai berikut :

1. Number of cities menunjukkan jumlah kota yang akan diselesaikan dalam TSP.
2. Distances between cities, dimana jika i j = x maka jarak dari kota i ke kota j adalah x.

Data ini masih berbentuk text yang nantinya dikonversikan ke dalam nilai-nilai tertentu.

Selain data dari berkas masukan seperti di atas, terdapat data masukan lain, yaitu berupa parameter Algoritma Genetika. Parameter-parameter tersebut adalah sebagai berikut :

- *Population Size* : merupakan angka yang menunjukkan jumlah kromosom pada suatu populasi dalam satu generasi

- *Generation* : angka yang menunjukkan jumlah generasi yang diproses
- *Crossover rate* : menunjukkan kemungkinan suatu kromosom dalam satu generasi untuk melakukan pindah silang.
- *Mutation rate* : menunjukkan kemungkinan suatu kromosom dalam satu generasi untuk melakukan mutasi.
- *Fitness Scaling* : di dalamnya berisi pilihan untuk sistem mekanisme penyesuaian yang digunakan. Ada tiga pilihan yaitu Windowing, Eksponensial, atau Normalisasi linier.
- *Crossover methodes* : berisi pilihan sistem pindah silang yang digunakan. Ada 3 pilihan yaitu PMX, OX, dan CX
- *Mutation methodes* : berisi pilihan sistem mutasi yang dipilih. Ada 3 pilihan yaitu Inversion, Insertion, dan Rexiprocal Exchange.
- *Linear Normalization Parameter* : berisi mengenai parameter untuk sistem mekanisme penyesuaian Normalisasi linier. Ada tiga parameter yang harus diisi, yaitu base, decrement, minimal.
- *Report* : jika masukan ini menunjuk angka x, maka tiap x generasi dilakukan pencatatan hasil.
- *Show best* : masukan ini menunjukkan berapa individu terbaik dari generasi yang ditunjuk oleh angka report untuk dicatat.

4.1.2 Data Saat Pemrosesan

Data yang diperlukan selama Algoritma Genetika berlangsung adalah data dari populasi, yang mana populasi ini berisi kromosom yang mewakili sebuah lintasan. Data ini diproses dan diganti selama iterasi dalam proses Algoritma Genetika. Untuk data yang pertama maka dilakukan inisialisasi populasi awal dengan dipandu bilangan acak murni untuk kromosom pertama, karena untuk menghasilkan lintasan yang valid. Untuk kromosom selanjutnya, maka digunakan permutasi Josephus seperti yang telah dibahas dalam Bab III. Data ini disimpan dalam struktur data berupa array of list.

4.1.3 Data Keluaran

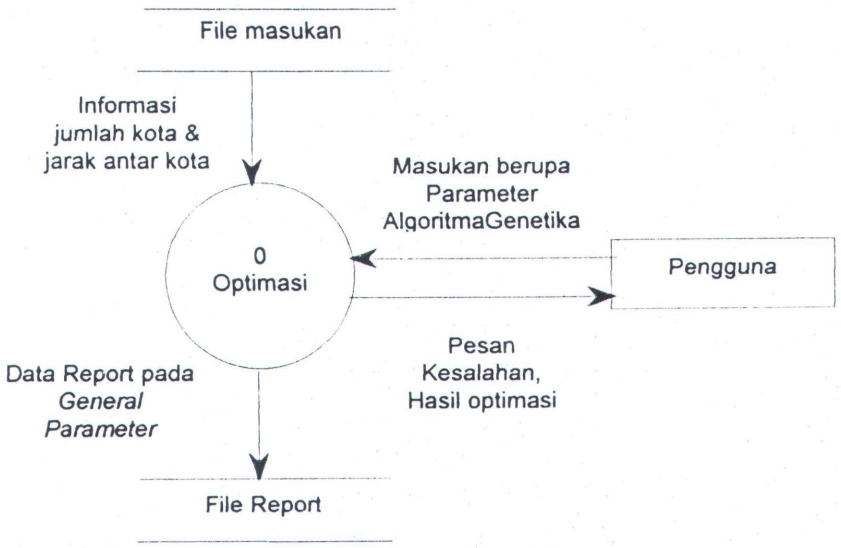
Data keluaran yang dihasilkan adalah nilai terbaik yang dihasilkan dari proses Algoritma Genetika yaitu jarak terpendek yang mana data ini dinyatakan dengan bilangan real. Data lain untuk data keluaran ini adalah kromosom yang terbaik dengan kata lain yaitu lintasan dengan jarak terpendek. Seperti halnya kromosom-kromosom yang lain lintasan ini dinyatakan dalam bentuk list.

4.2 PERANCANGAN PROSES

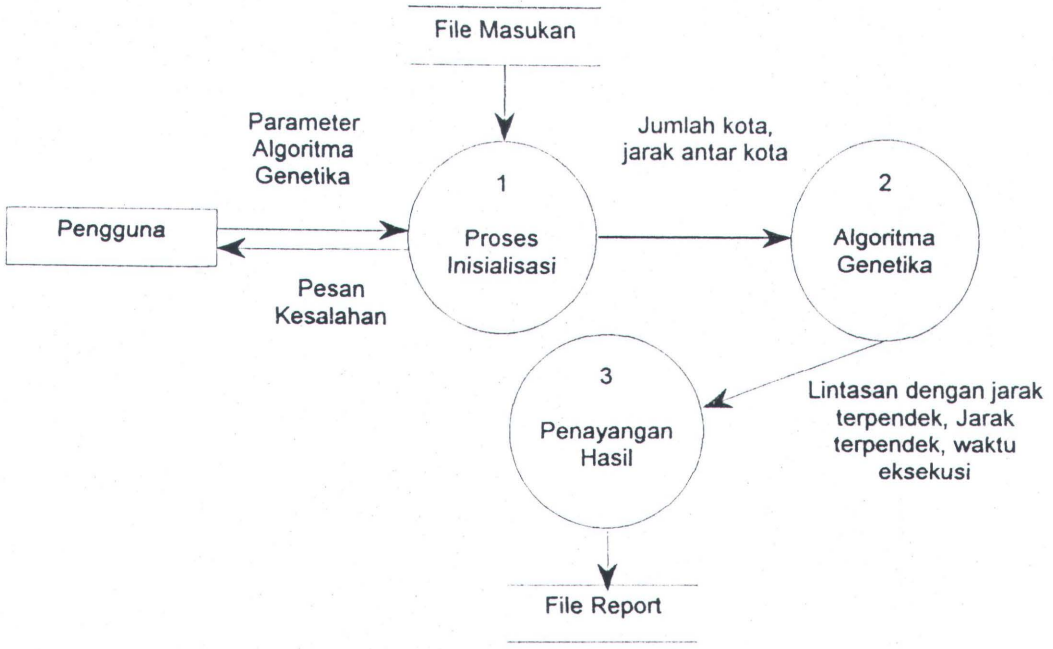
Secara global, pemrogaman perangkat lunak ini dibagi dalam tiga bagian besar. Proses yang pertama adalah inisialisasi dari masukan yang telah diberikan, sehingga didapatkan angka-angka untuk jumlah kota, jarak antar kota dan parameter-parameter yang dibutuhkan untuk proses Algoritma Genetika. Kemudian dari sini, data tersebut digunakan untuk proses Algoritma Genetika. Dari proses ini hasilnya akan ditampilkan pada antarmuka keluaran. Untuk

menguraikan rancangan proses, maka digunakan Diagram Alir Data dan Hirarki Modul seperti di bawah ini.

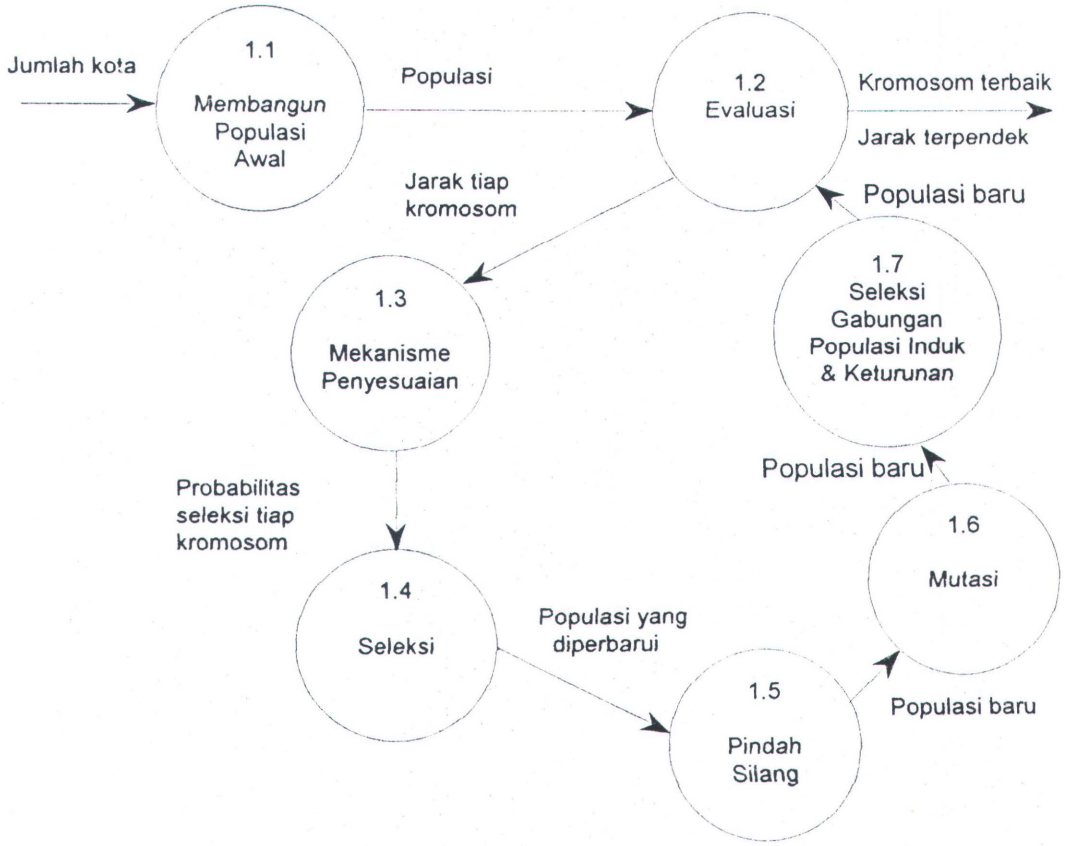
Susunan Diagram Alir Data untuk perangkat lunak yang dibuat adalah sebagaimana yang digambarkan pada Gambar 4.1 sampai dengan Gambar 4.4, diawali dengan Diagram Alir Data Level 0 sampai dengan Level 3.



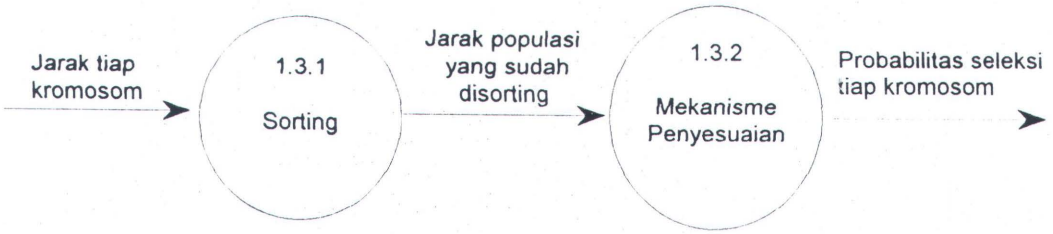
Gambar 4.1 Diagram Alir Data Level 0



Gambar 4.2 Diagram Alir Data Level 1

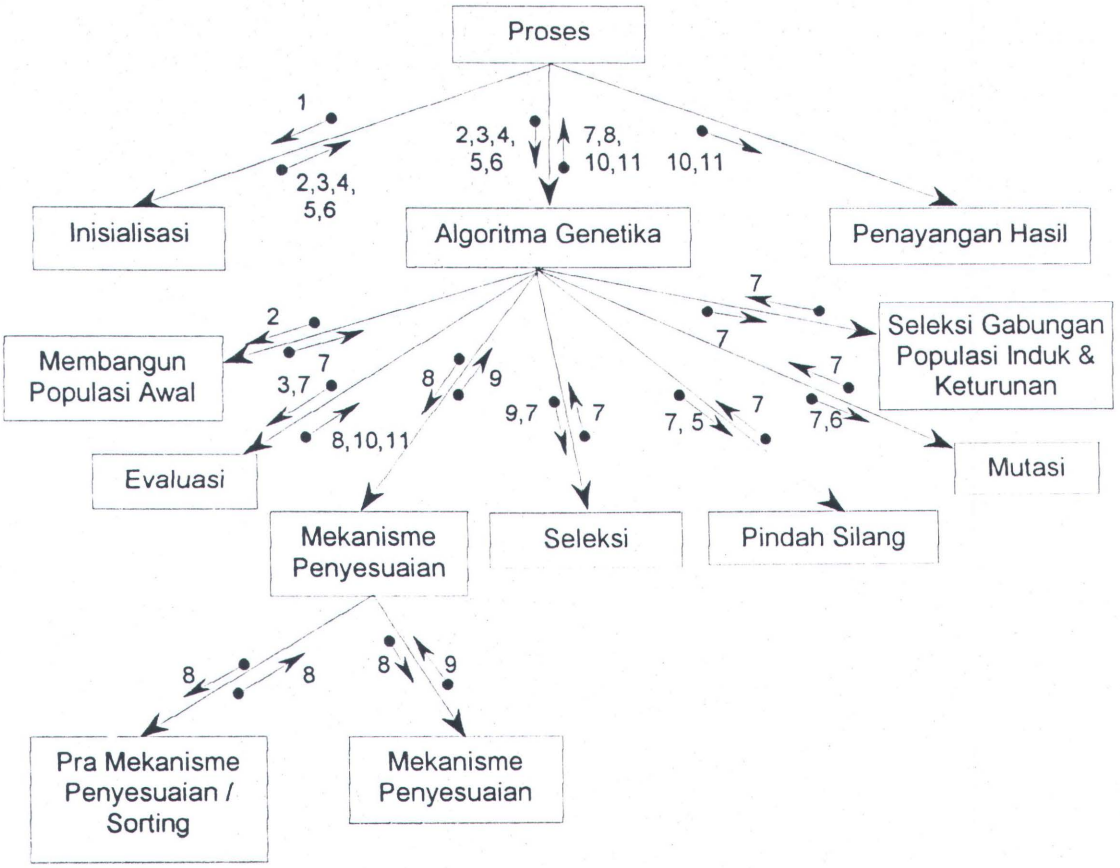


Gambar 4.3 Diagram Alir Data Level 2, Proses Algoritma Genetika



Gambar 4.4 Diagram Alir Data Level 3, Proses Mekanisme Penyesuaian

Hirarki Modul yang diimplementasikan pada Tugas Akhir ini secara keseluruhan dapat dijabarkan seperti Gambar 4.5.



Gambar 4.5 Hirarki Modul dari Keseluruhan Tugas Akhir

Nomor-nomor pada Hirarki Modul tersebut mewakili data-data yang masuk dan keluar sesuai dengan DAD pada Gambar 4.1 sampai Gambar 4.5, dengan uraian sebagai berikut:

1 Data berbentuk text dari file *.txt

2 Jumlah kota

3 Jarak antar kota

4 Parameter jumlah individu dalam populasi

5 Parameter probabilitas pindah silang

6 Parameter probabilitas mutasi

7 Populasi, baik populasi lama ataupun populasi yang sudah diperbarui

8 Jarak tiap kromosom

9 Probabilitas seleksi dari tiap kromosom

10 Jarak terpendek

11 Kromosom terbaik/lintasan terpendek

Dari DAD dan Hirarki Modul tersebut atas, maka dapat dijabarkan proses-prosesnya sebagai berikut :

- Membangun populasi awal, dimana di sini dilakukan dengan dipandu bilangan acak murni untuk kromosom pertama dan kromosom selanjutnya dengan Permutasi Josephus.
- Proses evaluasi. Pada bagian ini ada dua sub proses yaitu proses menghitung jarak lintasan dari populasi yang telah dihasilkan dan proses selanjutnya adalah mencatat lintasan terpendek. Sub proses yang kedua tidak berpengaruh terhadap proses selanjutnya (setelah proses evaluasi). Dari sub proses ini iterasi Algoritma Genetika akan berhenti apabila iterasi telah dilakukan sebanyak jumlah generasi.

- Mekanisme Penyesuaian, di mana bagian ini juga terdiri dari dua sub proses, yaitu proses persiapan untuk penyesuaian berupa proses sorting dan kemudian dilanjutkan dengan proses penyesuaian, hal ini dilakukan untuk semua metode, baik metode windowing, eksponensial, maupun normalisasi linier seperti yang telah dijelaskan pada Bab III.
- Proses Seleksi. Pada bagian ini dilakukan seleksi campuran(stokastik dan elitism) dari populasi lama hingga didapatkan populasi baru.
- Proses Pindah Silang. Proses ini dilakukan sebanyak p_c * *pop_size*, dengan salah satu metode pindah silang untuk permasalahan kombinatorial. *Pop_size* adalah jumlah populasi dalam setiap generasi.
- Proses Mutasi. Proses ini dilakukan sebanyak p_m * *pop_size* * L, dimana L adalah jumlah kota dalam lintasan (jumlah gen dalam setiap kromosom). Dari proses pindah silang dan mutasi dihasilkan populasi keturunan.
- Proses Seleksi Gabungan Populasi Induk dan Populasi Keturunan. Pada bagian ini dilakukan penggabungan populasi induk (populasi lama sebelum dilakukan proses seleksi) dengan populasi keturunan (populasi yang dihasilkan setelah pindah silang dan mutasi). Kemudian dari populasi gabungan dengan kapasitas 2 * *pop_size* ini dilakukan seleksi terhadap kromosom-kromosom terbaik untuk populasi baru, yang mana populasi baru ini ukurannya kembali lagi sebanyak *pop_size* kromosom. Apabila iterasi belum selesai (belum sebanyak jumlah generasi), maka proses akan kembali ke proses evaluasi dan seterusnya, dengan populasi yang dievaluasi adalah populasi baru yang didapatkan pada bagian ini.
- Proses Penayangan Hasil. Jika iterasi telah dilakukan sebanyak jumlah generasi, maka proses Algoritma Gentika akan berhaenti. Selanjutnya pada

bagian ini dilakukan penayangan hasil optimasi berupa jarak terpendek dan lintasan terpendek yang didapatkan.

4.3 PERANCANGAN TAMPILAN

Tampilan (sebagai fasilitas interaksi dengan user) dirancang dalam 2 bentuk, yaitu tampilan masukan dan keluaran. Tampilan masukan bertanggung jawab untuk menangani data masukan, yang meliputi persoalan yang akan dioptimasi dan parameter-parameter yang digunakan untuk proses optimasi. Di samping form untuk tampilan parameter ini, ada form lain yang menampilkan file *.txt yang dibuka oleh pengguna. Tampilan keluaran bertanggung jawab untuk menampilkan hasil optimasi.

4.3.1 Tampilan Masukan

Tampilan masukan dibagi dalam 4 (tiga) bagian, yaitu:

- Bagian *General*, yang berisikan parameter TSP dan parameter yang menentukan bagaimana hasil optimasi ditampilkan.
- Bagian *GA's Parameters*, berisikan parameter-parameter optimasi Algoritma Genetika
- Bagian *Distances between cities*, menampilkan tabel yang berisi jarak antar kota.
- Bagian Branch and Bound, menampilkan awal kota dicarinya lintasan pada metode Branch and Bound.

4.3.2 Tampilan Keluaran

Tampilan keluaran dibagi menjadi 3 (tiga) bagian, yaitu;

- Bagian *Solution*, menunjukkan hasil optimasi yang diperoleh Algoritma Genetika dan Metode Branch and Bound. Hasil optimasi meliputi sebuah lintasan terbaik, dengan urutan kota-kotanya dan waktu yang dibutuhkan untuk proses optimasi.
- Bagian *GA's Performance*, menunjukkan proses optimasi Algoritma Genetika dari generasi ke generasi.
- Bagian *Generation List*, menunjukkan keadaan populasi dalam generasi-generasi tertentu.

4.4 IMPLEMENTASI PADA BAHASA PEMROGRAMAN

Untuk mengaplikasikan perancangan perangkat lunak dalam sebuah bahasa pemrograman, maka pada bagian ini dijelaskan mengenai implementasi dari struktur data yang digunakan dan kelas data utama dan rutin-rutin penting yang digunakan.

4.4.1 Struktur Data

Setelah file *.txt dibuka, maka didapatkan informasi mengenai jumlah kota dan jarak antar kota yang masih berbentuk text. Dari sini dengan proses inisialisasi data dikonversikan ke nilai *word* untuk jumlah kota, dan *real* untuk jarak antar kota. Jarak antar kota secara keseluruhan disimpan dalam sebuah array dinamik yang dibangun berdasarkan jumlah kota. Struktur data untuk informasi ini dideklarasikan seperti di bawah ini:

```
numcity : word;
distances : array of array of real;
```

Untuk kromosom-kromosom yang dibangun, maka struktur data yang digunakan adalah list dengan membuat kelas data sendiri yaitu TMyList, yang mana kelas data ini akan dijelaskan pada subbab berikutnya. Sedangkan populasi yang akan dibangun diimplementasikan dengan struktur data berbentuk array dinamik yang dibangun berdasarkan parameter *Population Size*. Deklarasi dari struktur tersebut adalah sebagai berikut:

```
Pop : Array of TMyList
```

Untuk hasil berupa jarak terpendek dinyatakan dengan tipe data real. Sedangkan untuk lintasan terbaik, seperti lintasan-lintasan lain maka digunakan tipe data yang diambil dari kelas TMyList.

4.4.2 Kelas Data

Untuk membangun kromosom-kromosom yang ada dalam populasi, maka dibangun sebuah kelas yaitu kelas TMyList. Kelas ini membangun list dengan cara kerja yang hampir sama dengan kelas TList pada bahasa pemrograman Delphi. Hanya saja kapasitasnya diperbesar dengan ukuran nilai positif dari integer, yaitu 2.147.483.647 list. Struktur kelas tersebut adalah sebagai berikut

```
Type TValue         =  word;  // Type data yang dimasukkan list
     TMyList        = Class(TObject)
         Private
            iCount  : Longint;     // Max jumlah items = MaxInt =
            2.147.483.647 atau batas memory available
            iList   : TMemoryStream;
            iSize   : Word;
         Protected
            Procedure WItems(x : Longint; Value : TValue);
            Function RItems(x : Longint) : TValue;
            Destructor Destroy; Override;
         Public
            Constructor Create;
            Procedure Free;
            Procedure Clear;
            Procedure Add(Value : TValue);
            Procedure Delete(x : Longint);
```

```
        Property Items[x : Longint] : TValue Read RItems Write
        WItems; Default;
        Property Count : Longint Read iCount;
End;
```

Selain itu digunakan suatu kelas tambahan, yang mana kelas tersebut berfungsi untuk membangkitkan bilangan acak dengan distribusi uniform. Struktur kelas tersebut bersama rutin pentingnya adalah sebagai berikut:

```
const
    IM1 = 2147483563;
    IM2 = 2147483399;
    AM = 1/IM1;
    IMM1 = IM1-1;
    IA1 = 40014;
    IA2 = 40692;
    IQ1 = 53668;
    IQ2 = 52774;
    IR1 = 12211;
    IR2 = 3791;
    NTAB = 32;
    NDIV = 1 + IMM1 div NTAB;
    EPS = 1.2 * 0.000911881965554517;
    RNM = 1.0 - EPS;

type
    TRandom = Class (TObject)
    private
    public
       Function RandDev (var idum : integer) : real;
    end;

Function TRandom.Randdev (var idum : integer) : real;
var
   j,k, idum2, iy : integer;
   iv : array[0..NTAB-1] of integer;
   temp : real;
begin
   idum2 := 123456789;
   iy := 0;
   if idum <= 0 then
   begin
      if (-(idum)<1) then idum := 1
         else idum := -(idum);
      idum2 := idum;
      j := NTAB + 7;
      while j <= 0 do
      begin
         k := idum div IQ1;
         idum := IA1 * (idum-k*IQ1)-k*IR1;
         if idum < 0 then idum := idum + IM1;
         if j<NTAB then iv[j] := idum;
         dec(j)
```

```
      end;
      iy := iv[0];
    end;
    k := idum div IQ1;
    idum := IA1 * (idum-k*IQ1)-k*IR1;
    if idum<0 then idum := idum + IM1;
    k := idum2 div IQ2;
    idum2 := IA2 * (idum2-k*IQ2)-k*IR2;
    if idum2<0 then idum2 := idum2 + IM2;
    j := iy div NDIV;
    iy := iv[j]-idum2;
    iv[j] := idum;
    if iy < 1 then iy := iy + IMM1;
    temp := AM * iy;
    if (temp  > RNM) then result := RNM
    else result := temp;
end;
```

Untuk kelas utama yang melakukan proses Algoritma Genetika, maka dideklarasikan seperti di bawah ini:

```
type
  TGAs_Process = Class (Tobject)
  private
  { Private declarations }
    Function show (n : TMyList): string;
    Function Cek_random1 : TMyList;
    Function Cek_random2 (rand : TMyList; cek : word): Boolean;
    Procedure Inisial;
    Function Validity (checked : array of byte) : Boolean;
    Procedure Sort(flag : array of real; var A: array of byte);
    Procedure First_Pop;
    Procedure Testing(var distances : array of real;
                                  path : array of TMyList; hi : word);
    Procedure QuickSort(var distances : array of real;
        var path : array of TMyList; iLo, iHi: integer);
    Procedure LinearNorm;
    Procedure Windowing;
    Procedure Eksponensial;
    Procedure  RouletteWheel(lo:byte;  var  flagpop  :  array  of
            TMyList);
    Function FindCity(getpop : TMyList; city : word): word;
    Procedure PMX(flagpop : array of TMyList; var newpop,newpop1 :
            TMyList; p1,p2 : word);
    Procedure Cycle(flagpop : array of TMyList; var newpop,newpop1 :
            TMyList; p1,p2 : word);
    Procedure Order(flagpop : array of TMyList; var newpop,newpop1 :
            TMyList; p1,p2 : word);
    Procedure Report(gen : word; var time2 : real; fitness : real);
    Procedure Crossover(var flagpop : array of TMyList);
    Procedure  Insertion(numMut  :  word;  var  flagpop  :  array  of
            TMyList);
```

```
    Procedure Inversion(numMut : word; var flagpop : array of
            TMyList);
    Procedure REM(numMut : word; var flagpop : array of TMyList);
    Procedure Mutation(var flagpop : array of TMyList);
    Procedure Unity(flagpop : array of TMyList);

  public
  { Public declarations }
    Procedure Genetic;
  end;
```

4.4.3 Implementasi Rutin-Rutin

Beberapa rutin penting dalam implementasi proses Algoritma Genetika dijabarkan sebagai berikut.

- Rutin yang membangun populasi awal

```
{Create Initial Population with Josephus Permutation}
Procedure First_Pop;
var
   i,j,s,k,l,m : word;
   Sharer : TMyList;
   ARecord : TValue;
   flag : Array of real;
   PopFlag, FlagPop : Array of byte;
Begin
   setlength(flag,num);
   setlength(PopFlag,num);
   Pop[0] := TMyList.Create;
   repeat
   for i := 0 to num-1 do
   begin
      flag[i] := rand.RandDev(initseed);
      PopFlag[i] := i + 1;
   end;
   sort(flag,PopFlag)
   until Validity(PopFlag) = True;
   for m := 0 to num-1 do
       Pop[0].add(PopFlag[m]);

   sharer := Cek_Random1;
   setlength(FlagPop,num);
   for i := 1 to PopSize-1 do
   begin
      Repeat
         s := round(rand.RandDev(initseed) * (num-2)) + 1
      until (Cek_Random2(sharer, s) = True);
      j := round(rand.RandDev(initseed) * (num-1)) + 1;
      Pop[i] := TMylist.Create;
      for k := 0 to num-1 do
      begin
```

```
            ARecord := j;
            FlagPop[k] := ARecord;
            for l := 1 to s do
               begin
                  inc(j);
                  if j>num then j := 1
               end;
         end;
         if validity(FlagPop) = false then
            for k := 0 to num-1 do
                Pop[i].add(PopFlag[k]) else
            for k := 0 to num-1 do
                Pop[i].add(FlagPop[k]);
      end;
      sharer.free
end;
```

- Rutin yang melakukan seleksi Roulette Wheel

```
Procedure RouletteWheel(lo : byte);
var
   i,j,k,l,m,n : word;
   prob, qom : array of real;
   total, temp, randl : real;
   flagpop : array of TMylist;
begin
(* generate new population *)
   if total = 0 then
      for i := 0 to PopSize - 1 do
         begin
            flagpop[i] := TMyList.create;
            for n := 0 to num-1 do
                flagpop[i].add(pop[i].items[n])
         end else
            begin
               for j := 0 to PopSize-1 do
               begin
                  prob[j] := fit[j]/total;
                  temp := temp + prob[j];
                  qom[j] := temp
               end;
               for k := lo to PopSize-1 do
               begin
                  randl := rand.RandDev(initseed);
                  if randl<=qom[0] then
                  begin
                     flagpop[k] := TMyList.create;
                     for n := 0 to num-1 do
                        flagpop[k].add(pop[0].items[n])
                  end;
                  for l := 1 to PopSize-1 do
                     if (randl>qom[l-1]) and (randl<=qom[l]) then
                     begin
                        flagpop[k] := TMyList.create;
                        for n := 0 to num-1 do
                           flagpop[k].add(pop[l].items[n]);
```

```
                        break
                    end;
                end;
    end;
end;
```

- Rutin yang melakukan operasi tukar silang adalah sebagai berikut:

```
Procedure Crossover(var flagpop: array of TMyList);
var
    randl, cek : Array of word;
    newpop, newpop1 : TMyList;
    numcross, p1, p2 : word;
    i,j,k,l : word;
begin
  for i := 0 to numcross-1 do
    begin
       repeat
          randl[i] := round(rand.RandDev(initseed) * (PopSize-1))
       until ( cek[randl[i]] <> 1 );
       cek[randl[i]] := 1;
    end;
    j := 0;
    while j <= numcross-1 do
    begin
       p1 := randl[j];
       p2 := randl[j+1];
       if flagpop[p1] = flagpop[p2] then j := j+2
       else
          begin
          newpop := TMyList.create;
          newpop1 := TMyList.create;
          for k := 0 to num-1 do
          begin
             newpop.Add(flagpop[p1].items[k]);
             newpop1.Add(flagpop[p2].items[k]);
          end;
         Case TSPAppForm.Crossover.ItemIndex of
           0 : PMX(flagpop,newpop,newpop1,p1,p2);
           1 : Order(flagpop,newpop,newpop1,p1,p2);
           2 : Cycle(flagpop,newpop,newpop1,p1,p2);
          end;
          flagpop[p1].free;
          flagpop[p2].free;
          flagpop[p1] := TMyList.Create;
          flagpop[p2] := TMyList.Create;
          for l := 0 to num-1 do
          begin
             flagpop[p1].add(newpop.Items[l]);
             flagpop[p2].add(newpop1.Items[l]);
          end;
          newpop.free;
          newpop1.Free;
          j := j+2
       end;
end;
```

PMX Crossover

```
Procedure PMX(flagpop : array of TMyList; var newpop,newpop1 :
TMyList; p1,p2 : word);
var
   j, k, n, s : word;
   ARecord, ARecord1, ARecord2, ARecord3 : TValue;
begin
   j := round(rand.RandDev(initseed)* (num-2));
   repeat
      k := round(rand.RandDev(initseed) * (num-1))
   until  k > j;
   for n := j to k do
   begin
      ARecord := flagpop[p1].items[n];
      ARecord1 := flagpop[p2].items[n];
      if ARecord <> ARecord1 then
      begin
         s := FindCity(newpop1, ARecord);
         ARecord2 := newpop1.items[n];
         newpop1.items[n] := newpop1.items[s];
         newpop1.items[s] := ARecord2;

         s := FindCity(newpop, ARecord1);
         ARecord3 := newpop.items[n];
         newpop.Items[n] := newpop.Items[s];
         newpop.Items[s] := ARecord3;
      end;
   end;
end;
```

Cyclic Crossover

```
Procedure Cycle(flagpop : array of TMyList; var newpop,newpop1 :
TMyList; p1,p2 : word);
var
   j, t : word;
   ARecord, ARecord1 : TValue;
Begin
   j := round(rand.Randdev(initseed) * (num-1));
   t := flagpop[p1].items[j];

   repeat
      newpop.Items[j] := flagpop[p2].items[j];
      newpop1.Items[j] := flagpop[p1].items[j];
      ARecord1 := newpop.Items[j];
      j := FindCity(flagpop[p1],ARecord1)
   until ARecord1 = t;
end;
```

Order Crossover

```
Procedure Order(var newpop,newpop1 : TMyList; p1,p2 : word);
var
   j, k, n, s, l, i : word;
   vp : TMyList;
   ARecord, ARecord1 : word;
Begin
   j := round(rand.RandDev(initseed) * (num-2));
   repeat
      k := round(rand.RandDev(initseed) * (num-1))
   until k > j;

   if ((j = 0) and (k = num-1)) then
   begin
      vp := newpop;
      newpop := newpop1;
      newpop1 := vp
   end else

   if k = num-1 then n := 0 else n := k + 1;

   (* Shift and fill *)
   repeat
      while (True) do
      begin
         ARecord := newpop.Items[n];
         s := FindCity(flagpop[p2], ARecord);
         if ((s<j) or (s>k)) then break;

         (* shift members *)
         if n = num-1 then l := 0
           else l := n + 1;

         while (True) do
         begin
            if l=0 then newpop.Items[num-1] := newpop.Items[0]
              else newpop.Items[l-1] := newpop.Items[l];

            if l = k then break;

            if l = num-1 then l := 0
              else inc(l)
         end;
      end;

      while (True) do
      begin
         ARecord1 := newpop1.Items[n];
         s := FindCity(flagpop[p1],ARecord1) ;

         if ((s<j) or (s>k)) then break;

         (* shift members *)
         if n = num-1 then l := 0
           else l := n + 1;
```

```
            while (True) do
            begin
               if l=0 then newpop1.Items[num-1] := newpop1.Items[0]
                  else newpop1.Items[l-1] := newpop1.Items[l];

               if l = k then break;
               if l = num-1 then l := 0
                  else inc(l)
            end;
         end;
         if n = num-1 then n := 0 else inc(n);
      until n = j;
      for n  := j to k do
      begin
         newpop.Items[n] := flagpop[p2].items[n];
         newpop1.Items[n] := flagpop[p1].items[n]
      end;
end;
```

- Rutin yang melakukan operasi mutasi

```
Procedure Mutation(var Flagpop : array of TMyList);
var
   numMut : word;
begin
   numMut  :=  round  (StrToInt(TSPAppForm.Edit8.Text)/100  *  num  *
PopSize);
   if numMut < 1 then exit;
   case TSPAppForm.Mutation.ItemIndex of
   0 : Inversion(numMut, flagpop);
   1 : Insertion(numMut, flagpop);
   2 : REM(numMut, flagpop);
   end;
end;
```

Inversion Mutation

```
Procedure Inversion(numMut : word; var flagpop : array of TMyList);
var
   i : word;
   cek : Array of byte;
   rand1, p1, p2, temp : word;
begin
for i := 0 to numMut-1 do
   begin
      repeat
         rand1 := round(rand.RandDev(initseed)*(PopSize-1))
      until cek[rand1] <>1;
      cek[rand1] := 1;
      p1 := round(rand.RandDev(initseed)*(num-2));
      repeat
         p2 := round(rand.RandDev(initseed)*(num-1));
      until p1<p2;
      while p1<=p2 do
      begin
```

```
            temp := flagpop[rand1].items[p1];
            flagpop[rand1].items[p1] := flagpop[rand1].items[p2];
            flagpop[rand1].items[p2] := temp;
            inc(p1);
            dec(p2)
         end;
      end;
end;
```

Insertion mutation

```
Procedure Insertion(numMut : word; var flagpop : array of TMyList);
var
   i,j,k : word;
   rand1, MutPop, MutNum, NewPos, temp :word;
   cek : Array of byte;
begin
   setlength(cek,num*PopSize);
    for i := 0 to numMut-1 do
      begin
         Repeat
            rand1 := round(rand.RandDev(initseed)*(num * PopSize -
                     1)) +1;
         until cek[rand1] <> 1;
         cek[rand1] := 1;
         MutPop := rand1 div num;
         MutNum := rand1 mod num;
         Repeat
            NewPos := round(rand.RandDev(initseed)*(num-1));
         until NewPos <> MutNum;
         if NewPos < MutNum then
            begin
               temp := FlagPop[MutPop].items[MutNum];
               for j := MutNum-1 downto NewPos do
                  FlagPop[MutPop].items[j+1]:=
                  Flagpop[MutPop].items[j];
               FlagPop[MutPop].items[NewPos] := temp;
            end
         else
           begin
               temp := FlagPop[MutPop].items[MutNum];
               for k := MutNum to NewPos-1 do
                  FlagPop[MutPop].items[k]:=
                  FlagPop[MutPop].items[k+1];
               FlagPop[MutPop].items[NewPos] := temp;
           end;
      end;
end;
```

Rexiprocal Exchange Mutation

```
Procedure REM(numMut : word; var flagpop : array of TMyList);
var
   i : word;
   rand1, p1, p2, temp : word;
   cek : Array of byte;
```

```
begin
   While i < numMut do
   begin
      repeat
         rand1 := round(rand.RandDev(initseed)*(PopSize-1));
      until cek[rand1] <> 1;
      cek[rand1] := 1;
      p1 := round(rand.RandDev(initseed) * (num-1));
      repeat
         p2 := round(rand.RandDev(initseed) * (num-2))
      until p2<>p1;
      temp := flagpop[rand1].items[p1];
      flagpop[rand1].items[p1] := flagpop[rand1].items[p2];
      flagpop[rand1].items[p2] := temp;
      inc(i)
   end;
end;
```

BAB V
UJI COBA DAN PEMBAHASANNYA

BAB V

HASIL UJI COBA DAN PEMBAHASANNYA

Dalam bab ini diuraikan mengenai hasil uji coba perangkat lunak dengan masukan yang sama dengan perangkat lunak yang sudah pernah dibuat yang menggunakan metode lain. Selain itu hasil uji coba perangkat lunak ini akan dibandingkan juga dengan metode Branch and Bound yang dibuat perangkat lunaknya bersamaan dengan perancangan Tugas Akhir ini. Hal ini kami lakukan karena keterbatasan untuk mencari perangkat lunak yang sudah jadi dan juga dengan membuat sendiri maka dapat mempermudah penulis untuk melakukan perbandingan yang mendekati perancangan yang dibuat.

Selain itu, dari hasil uji coba yang telah dilakukan kami akan menganalisa apakah rancangan ini dapat memenuhi tujuan yang akan diraih seperti yang dipaparkan pada Bab I.

Kami juga akan melakukan ujicoba terhadap perangkat lunak dengan masukan parameter Algoritma Genetika yang berbeda, sehingga bisa dianalisa pengaruh dari parameter-parameter tersebut beserta perubahannya.

Sebelum melakukan ujicoba kami paparkan dulu sistem pendukung berupa spesifikasi perangkat lunak dan perangkat keras. Tugas Akhir ini dibuat dalam bahasa pemrograman *Borland Delphi 4.0*. Konfigurasi minimum komputer yang diperlukan agar perangkat lunak dapat bekerja secara layak, adalah sebagai berikut:

- Sistem operasi 32 bit

- Prosesor Pentium 200
- Memori 32 MB

5.1 UJI COBA DENGAN BERBAGAI NILAI UNTUK PARAMETER

Unjuk kerja Algoritma Genetika sangat dipengaruhi oleh parameter yang digunakan dan nilai-nilai yang dimasukkan terhadap parameter tersebut. Dalam Tabel 5.1 diperlihatkan hasil dengan berbagai kemungkinan parameter yang digunakan.

Dari hasil yang didapatkan dapat disimpulkan bahwa parameter-parameter yang berpengaruh terhadap kinerja Algoritma Genetika dalam menyelesaikan TSP agar didapatkan hasil yang optimal dengan terhindarnya kasus konvergensi dini adalah:

- Ukuran populasi. Parameter ini adalah parameter yang pertama kali berpengaruh terhadap kinerja Algoritma Genetika. Untuk jumlah kota yang besar, maka ukuran populasi untuk ruang pencarian harus diberikan dengan cukup. Namun permasalahan dengan jumlah kota yang relatif kecil, maka ukuran populasi cukup diberikan dengan ukuran yang kecil. Dari tabel tersebut dapat kita lihat untuk jumlah kota sebesar 7 cukup dengan ukuran populasi 11, maka kita sudah mendapatkan nilai yang optimum. Untuk jumlah kota 10 ternyata dengan ukuran populasi 30 menunjukkan hasil yang lebih meningkat dibanding dengan jumlah populasi 15. Namun jika ukuran populasi terlalu besar, akan percuma karena mengakibatkan pemborosan waktu. Hal ini dapat dilihat dalam uji coba untuk 15 kota dengan ukuran

populasi yang diperbesar (sampai 2000), tetap menghasilkan nilai optimum pada wilayah yang sama dengan jumlah populasi 1000.

- Parameter kedua yang menentukan adalah metode mekanisme penyesuaian yang digunakan. Mekanisme Linear Normalization relatif memberikan unjuk kerja yang lebih bagus dibanding metode yang lain.
- Parameter berikutnya yang berpengaruh adalah besar probabilitas pindah silang dan mutasi.
- Yang turut berpengaruh selanjutnya adalah metode mutasi yang digunakan. Karena fungsi operator mutasi terbukti adalah untuk menghindari Konvergensi Dini. Dalam kasus yang diangkat pada subbab 3.3 maka mutasi yang paling cocok digunakan adalah metode Inversion (metode yang lain tidak menghasilkan nilai yang paling optimum) .
- Untuk parameter-parameter lainnya seperti ukuran generasi, asalkan ukuran populasi yang diberikan cukup, parameter ini tidak banyak berpengaruh. Namun tidak menutup kemungkinan suatu saat dibutuhkan ukuran generasi yang cukup. Sedangkan untuk metode Pindah Silang, secara umum semua metode yang ada memberikan kinerja yang sama pada Algoritma Genetika. Sedangkan untuk jenis seleksi campuran (yang melibatkan elitism selection), relatif memberikan hasil yang lebih bagus.

Tabel 5.1 Tabel *Benchmarks* (Hasil Uji Coba dari Berbagai Parameter)

Jumlah kota	Σ Populasi	Generasi	Pindah Silang		Mutasi		Mekanisme Penyesuaian	Jika Normalisasi Linier			Hasil Algoritma Genetika			Hasil Branch & Bound		
			Rate (%)	Metode	Rate (%)	Metode		Base	Dec	Min	Jarak	Waktu	G	Jarak	Waktu	S
7 (7-3)	11	10	60	PMX	1	Invers	Windowing				3.018	0.00	10	3.018	0.05	3
	110	10	60	PMX	1	Invers	Windowing				2.939	0.29	6			
	11	10	60	OX	1	Invers	Windowing				3.151	0.11	7	2.832	0.05	7
	11	10	60	CX	1	Invers	Windowing				2.939	0.00	10	2.832	0.11	4
	110	10	60	CX	1	Invers	Windowing				2.939	0.28	9	3.136	0.06	1
	110	10	60	OX	1	Invers	Windowing				2.832	0.23	6	3.022	0.05	2
	11	10	80	PMX	5	Invers	Windowing				2.832	0.11	8	2.939	0.05	5
	11	10	80	OX	5	Invers	Windowing				3.022	0.06	3	2.832	0.06	6
	11	10	80	CX	5	Invers	Windowing				2.939	0.11	9			
	110	10	80	PMX	5	Invers	Windowing				2.832	0.25	10			
	110	10	80	OX	5	Invers	Windowing				2.832	0.3	6			
	110	10	80	CX	5	Invers	Windowing				2.832	0.34	10			
	110	100	90	PMX	10	Invers	Windowing				2.832	3.01	8			
	110	100	90	PMX	10	Insert	Windowing				2.832	3.10	10			
	110	100	90	PMX	10	REM	Windowing				2.832	3.22	6			
	11	10	60	PMX	1	Invers	Eksponensial				3.377	0.05	6			
	11	10	60	PMX	1	Invers	LN	100	10	1	3.151	0.11	9			
	11	10	60	PMX	1	Invers	LN	100	10	1	3.151	0.11	9			
	11	10	60	PMX	1	Invers	LN	100	10	1	**2.832**	**0.00**	**6**			
10 (10-4)	15	10	60	PMX	1	REM	Windowing				231	0.05	7	218	1.32	1
	30	10	60	PMX	1	REM	Windowing				213	0.11	9	218	1.37	2
	15	10	60	PMX	1	REM	Eksponensial				231	0.06	9	221	1.7	3
	15	10	60	OX	1	REM	Eksponensial				231	0.17	3	221	1.21	4
	15	10	60	CX	1	REM	Eksponensial				231	0.11	6	230	0.94	5

Keterangan : LN : Normalisasi Linier **Invers : Inversion** **Insert : Insertion**
G : Algoritma Genetika menemukan nilai optimum pada generasi ke-... **S : Kota awal dimulainya lintasan dalam metode Branch & Bound**

Tabel 5.1 Tabel *Benchmarks* (Lanjutan)

Jumlah kota	Σ Populasi	Generasi	Pindah Silang		Mutasi		Mekanisme Penyesuaian	Jika Normalisasi Linier			Hasil Algoritma Genetika			Hasil Branch & Bound		
			Rate (%)	Metode	Rate (%)	Metode		Base	Dec	Min	Jarak	Waktu	G	Jarak	Waktu	S
	30	10	60	PMX	1	REM	Eksponensial				217	0.45	6	220	0.82	6
	30	10	60	OX	1	REM	Eksponensial				217	0.16	6	214	0.88	7
	30	10	60	CX	1	REM	Eksponensial				214	0.11	8	226	1.81	9
	30	10	60	CX	1	Invers	Eksponensial				227	0.16	2	218	0.94	10
	30	10	60	CX	1	Insert	Eksponensial				221	0.16	1	225	0.83	8
	30	10	60	CX	1	REM	LN	100	10	1	230	0.110	7			
	30	10	60	CX	1	REM	LN	100	10	1	217	0.16	6			
	30	10	60	CX	1	REM	LN	100	10	1	217	0.11	6			
	30	10	60	CX	1	REM	LN	100	10	1	210	0.54	10			
	30	10	60	CX	1	REM	LN	100	10	1	**208**	**3.18**	**85**			
13 (13-4)	30	20	80	PMX	5	Insert	Eksponensial				395	0.39	9	344	14.94	3
	30	20	80		5	Insert	Windowing				284	0.23	19			
	50	20	80	OX	5	Insert	Windowing				242	0.38	19	431	45.54	1
	50	20	80	OX	5	Insert	LN	100	20	1	263	0.5	17	326	14.44	2
	50	20	80	OX	5	Insert	LN	100	30	1	249	0.59	12	346	0.99	4
	50	20	80	OX	5	Insert	LN	100	40	1	202	0.51	15	436	5.27	5
	100	20	80	OX	5	Insert	LN	100	40	1	239	0.83	12	382	10.27	6
	100	20	80	OX	5	Insert	LN	100	40	1				323	17.69	7
	100	100	80	OX	5	Insert	LN	100	40	1	**149**	**3.33**	**70**	375	2.25	8
	100	100	80	PMX	5	Insert	LN	100	40	1	184	3.17	34	265	4.07	9
15 (15-4)	15	15	80	OX	5	Invers	LN	100	10	1	330	0.950	13	353	22.740	3
	50	15	80	OX	5	Invers	LN	100	10	1	302	0.480	8	358	34.380	4
	100	100	80	OX	5	Invers	LN	100	10	1	259	2.320	22	464	33.010	6
	1000	100	80	OX	5	Invers	LN	100	10	1	249	5.34	85	299	219.92	7
	1000	200	80	OX	5	Invers	LN	100	10	1	**217**	**44.360**	**83**			
	2000	100	80	OX	5	Invers	LN	100	10	1	230	66.760	50			

5.2 HASIL UJI COBA DIBANDING DENGAN HASIL METODE LAIN

Berikut ini beberapa hasil uji coba yang dilakukan terhadap perangkat lunak dengan masukan yang sama dengan masukan yang diberikan pada penelitian yang sudah pernah dilakukan oleh mahasiswa Teknik Informatika, yaitu Tugas Akhir yang dibuat oleh Nico Artanto, yang berjudul Aplikasi Hopfield Memory untuk menyelesaikan TSP.[23]

Hasil yang didapatkan uji coba untuk 10 kota dengan data jarak antar kota seperti ditampilkan pada Gambar 5.1. Parameter yang digunakan adalah Pop_Size = 110, Generation =100, Mutation Rate = 5 %, Crossover Rate = 80 %, dengan metode pindah silang PMX, mutasi Inversion, mekanisme penyesuaian Linear Normalization (Base = 100, Decrement = 10, Minimal = 1), seleksi dengan Roullete Wheel dan Elitism (Campuran), didapatkan hasil seperti yang diperlihatkan pada Gambar 5.2.

Hasil uji coba masukan 7 kota dengan data jarak antar kota seperti ditunjukkan Gambar 5.3, dengan parameter Pop_Size = 110, Generation =100, Mutation Rate = 5 %. Crossover Rate = 80 %, dengan metode pindah silang OX, mutasi Inversion, mekanisme penyesuaian Linear Normalization dengan parameter base = 100, decrement = 10, minimal = 1, seleksi dengan Roullete Wheel dan Elitism (Campuran), didapatkan hasil seperti yang ditunjukkan pada Gambar 5.4.

---

[23] Nico Artanto. *Aplikasi Hopfield Memory untuk Penyelesaian Traveling Salesman Problem*, Jurusan Teknik Komputer Fakultas Teknologi Industri Institut Teknologi Sepuluh Nopember, Surabaya, 1995.

The Solving of Traveling Salesman Problem by Genetic Algorithm

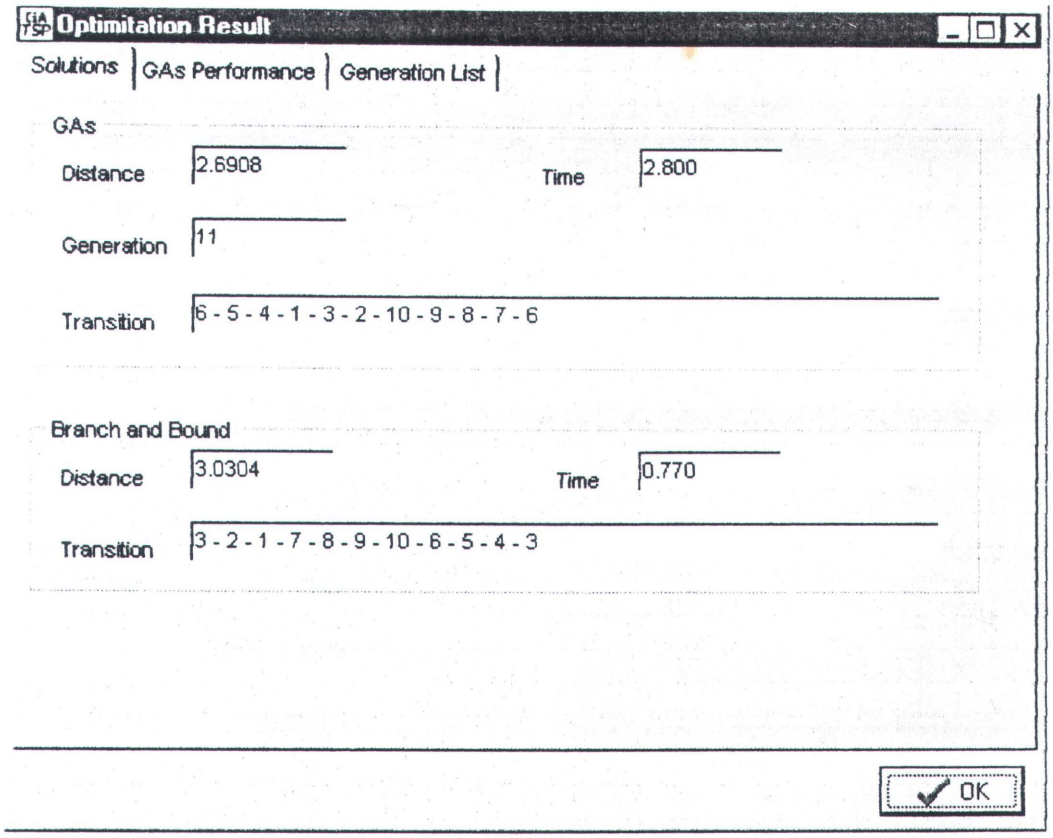
File Run Help

0% 0%

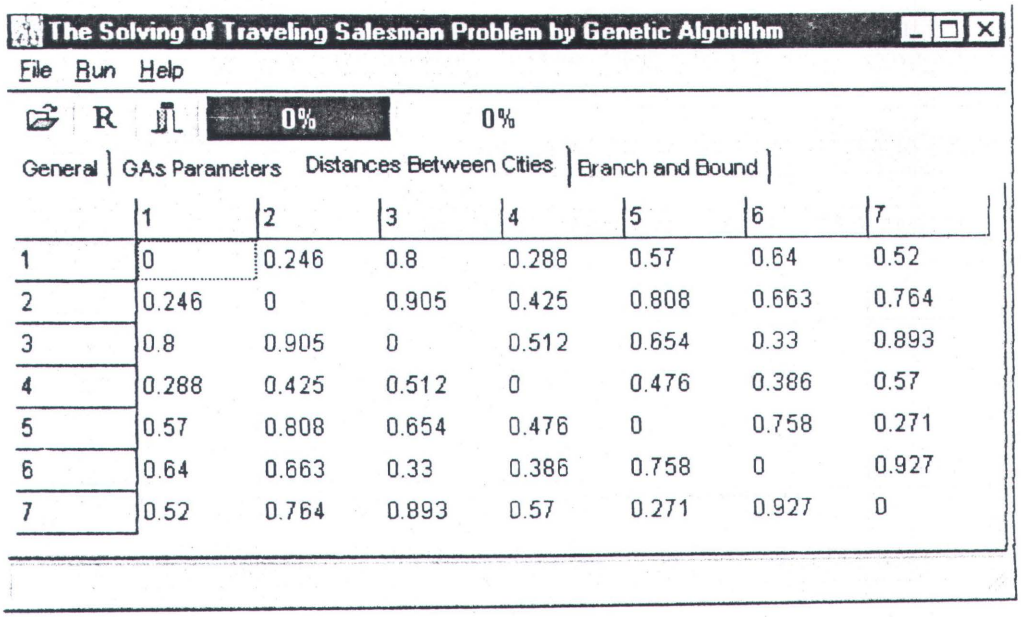
General | GAs Parameters | Distances Between Cities | Branch and Bound

	1	2	3	4	5	6	7	8	9	10
1	0	0.3361	0.3141	0.3601	0.5111	0.5176	0.2982	0.4564	0.3289	0.2842
2	0.3361	0	0.1107	0.6149	0.8407	0.8083	0.5815	0.6418	0.4378	0.3934
3	0.3141	0.1107	0	0.5349	0.7919	0.8207	0.5941	0.6908	0.4982	0.4501
4	0.3601	0.6149	0.5349	0	0.3397	0.6528	0.5171	0.7375	0.671	0.6323
5	0.5111	0.8407	0.7919	0.3397	0	0.4579	0.4529	0.6686	0.7042	0.6857
6	0.5176	0.8083	0.8207	0.6528	0.4579	0	0.2274	0.2937	0.4494	0.4654
7	0.2982	0.5815	0.5941	0.5171	0.4529	0.2274	0	0.2277	0.269	0.2674
8	0.4564	0.6418	0.6908	0.7375	0.6686	0.2937	0.2277	0	0.21	0.2492
9	0.3289	0.4378	0.4982	0.671	0.7042	0.4494	0.269	0.21	0	0.0498
10	0.2842	0.3934	0.4501	0.6323	0.6857	0.4654	0.2674	0.2492	0.0498	0

Gambar 5.1 Data jarak untuk uji coba 10 kota

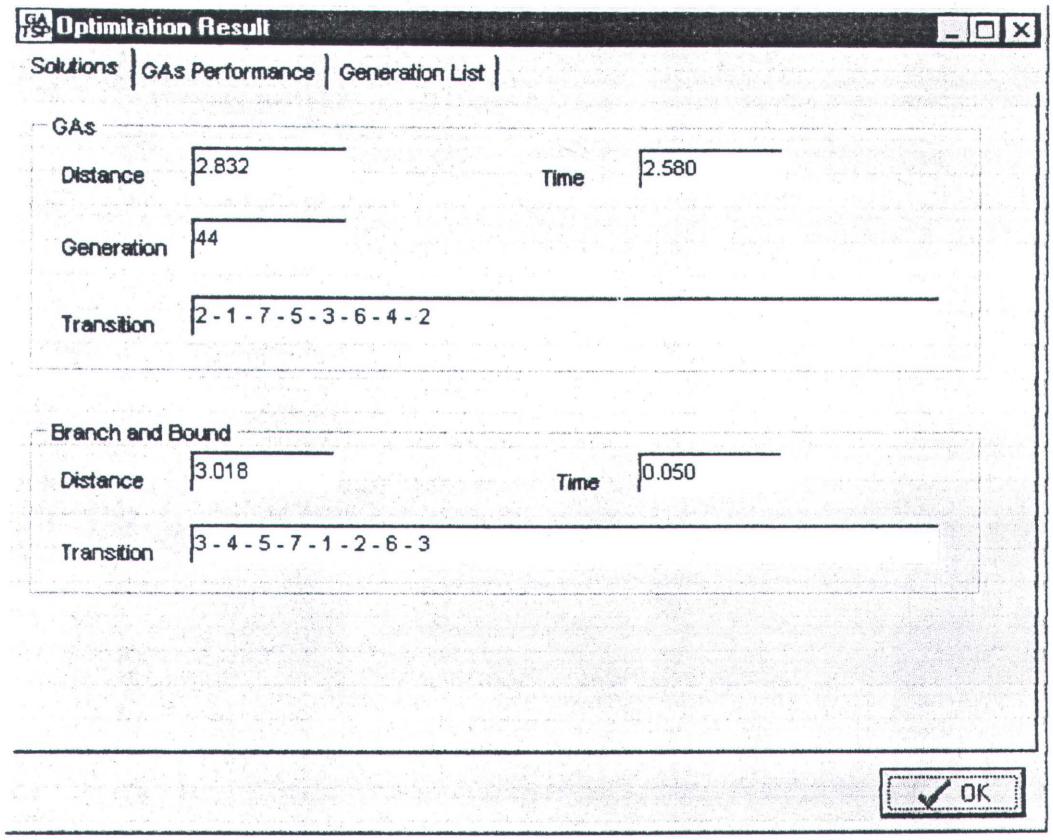


Gambar 5.2 Hasil Uji Coba untuk 10 Kota



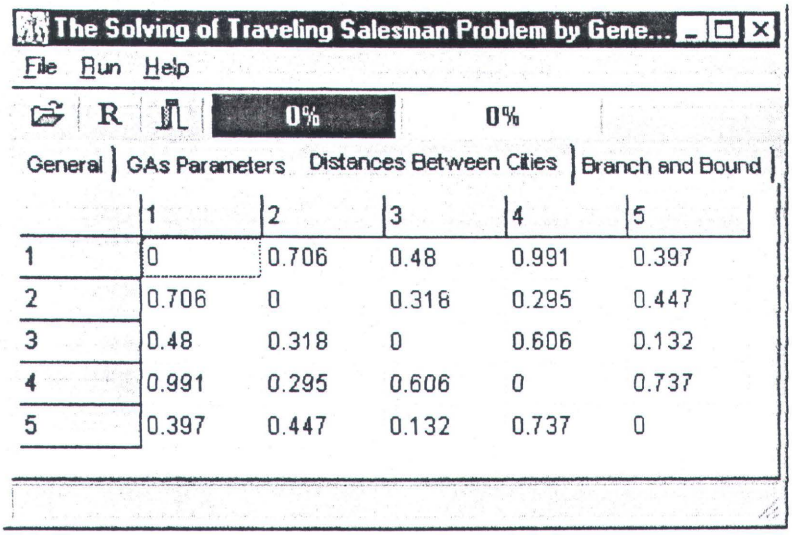
	1	2	3	4	5	6	7
1	0	0.246	0.8	0.288	0.57	0.64	0.52
2	0.246	0	0.905	0.425	0.808	0.663	0.764
3	0.8	0.905	0	0.512	0.654	0.33	0.893
4	0.288	0.425	0.512	0	0.476	0.386	0.57
5	0.57	0.808	0.654	0.476	0	0.758	0.271
6	0.64	0.663	0.33	0.386	0.758	0	0.927
7	0.52	0.764	0.893	0.57	0.271	0.927	0

Gambar 5.3 Data Jarak untuk Uji Coba 7 Kota



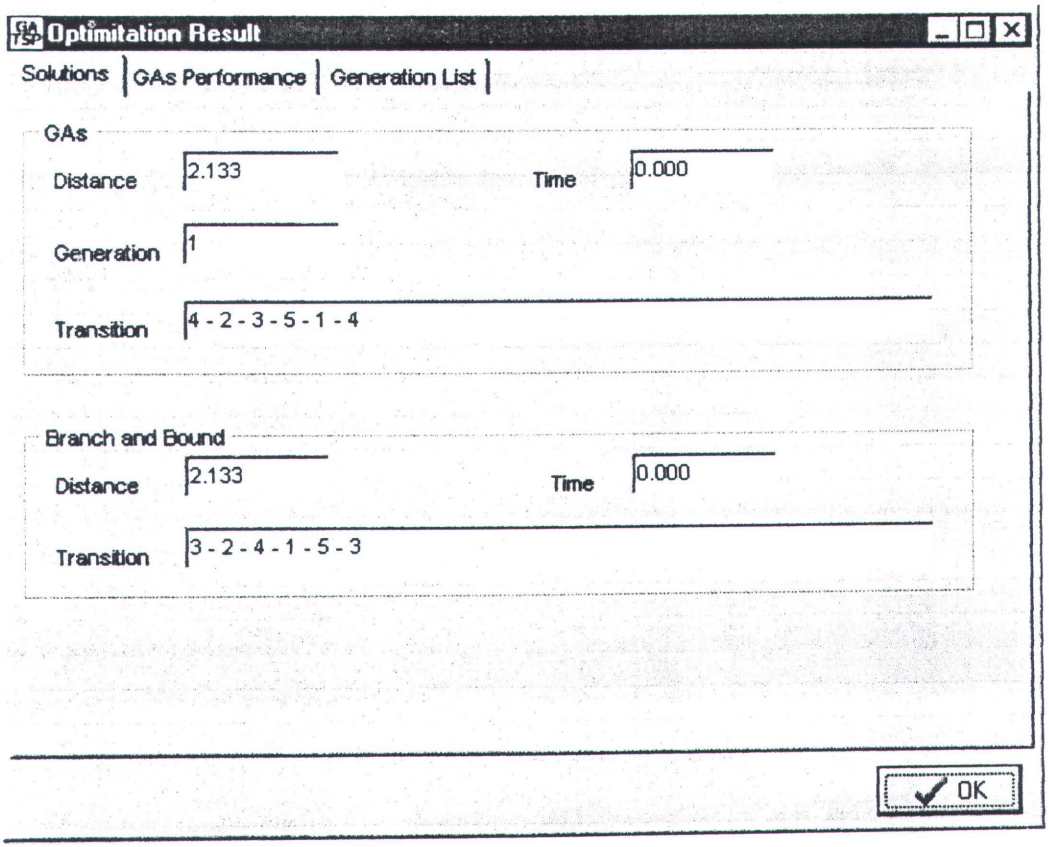
Gambar 5.4 Hasil Uji Coba untuk 7 Kota

Untuk hasil uji coba dengan masukan 5 kota dengan data jarak antar kota seperti yang ditunjukkan pada Gambar 5.5. Parameter yang digunakan adalah Pop_Size = 11, Generation =10, Mutation Rate = 5 %, Crossover Rate = 80 %, dengan metode pindah silang PMX, mutasi Insertion, mekanisme penyesuaian Windowing, seleksi dengan Roullete Wheel dan Elitism (Campuran), didapatkan hasil seperti yang diperlihatkan pada Gambar 5.6.



	1	2	3	4	5
1	0	0.706	0.48	0.991	0.397
2	0.706	0	0.318	0.295	0.447
3	0.48	0.318	0	0.606	0.132
4	0.991	0.295	0.606	0	0.737
5	0.397	0.447	0.132	0.737	0

Gambar 5.5 Data Jarak untuk Uji Coba 5 Kota



Gambar 5.6 Hasil Uji Coba untuk 5 Kota

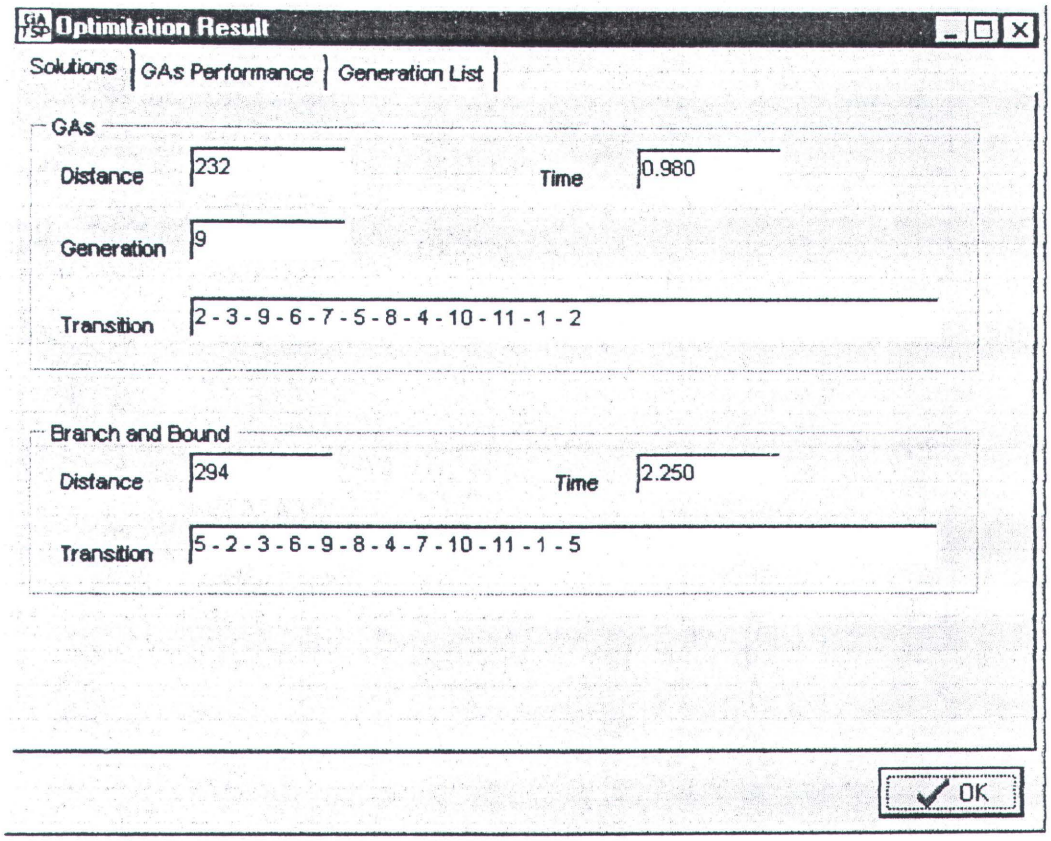
Untuk penyelesaian Hopfield Memory dengan uji coba 10 kota didapatkan jarak terpendek yaitu 2,8974 dengan lintasan terbaik 9–10–1–3–2–4–5–6–7–8–9. Sedangkan uji coba untuk 7 kota didapatkan jarak terpendek yaitu 3,167 dengan lintasan terbaik 3–7–5–4–1–2–6–3. Untuk uji coba 5 kota didapatkan jarak terpendek 2,133 dengan lintasan terbaik 4–1–2–5–3–4.

Hasil yang bisa dibandingkan terbatas hanya dengan data seperti di atas, karena dalam laporannya, panelitian yang sudah ada hanya menampilkan data 10, 7, dan 5 kota.

Dari hasil tersebut dapat disimpulkan bahwa Algoritma Genetika memberikan unjuk kerja yang lebih bagus dibanding dengan Hopfield Memory. Hal ini ditandai dengan hasil yang lebih optimum dibandingkan dengan hasil yang didapat Hopfield Memory. Disamping itu dalam uji cobanya Hopfield Memory belum tentu memberikan lintasan yang valid. Sedangkan untuk Algoritma Genetika selalu menghasilkan lintasan yang valid.

Sedangkan dari beberapa kali uji coba dibandingkan dengan metode Branch and Bound, untuk penyelesaian dengan jumlah kota yang kecil Algoritma Genetika kurang dari sisi kecepatan. Namun jika jumlah kota sedikit lebih besar (lebih dari 10), maka Algoritma Genetika unggul dari sisi kecepatan. Hal ini bisa dilihat pada hasil uji coba untuk 11 kota dan 13 kota pada Gambar 5.7 dan Gambar 5.8.

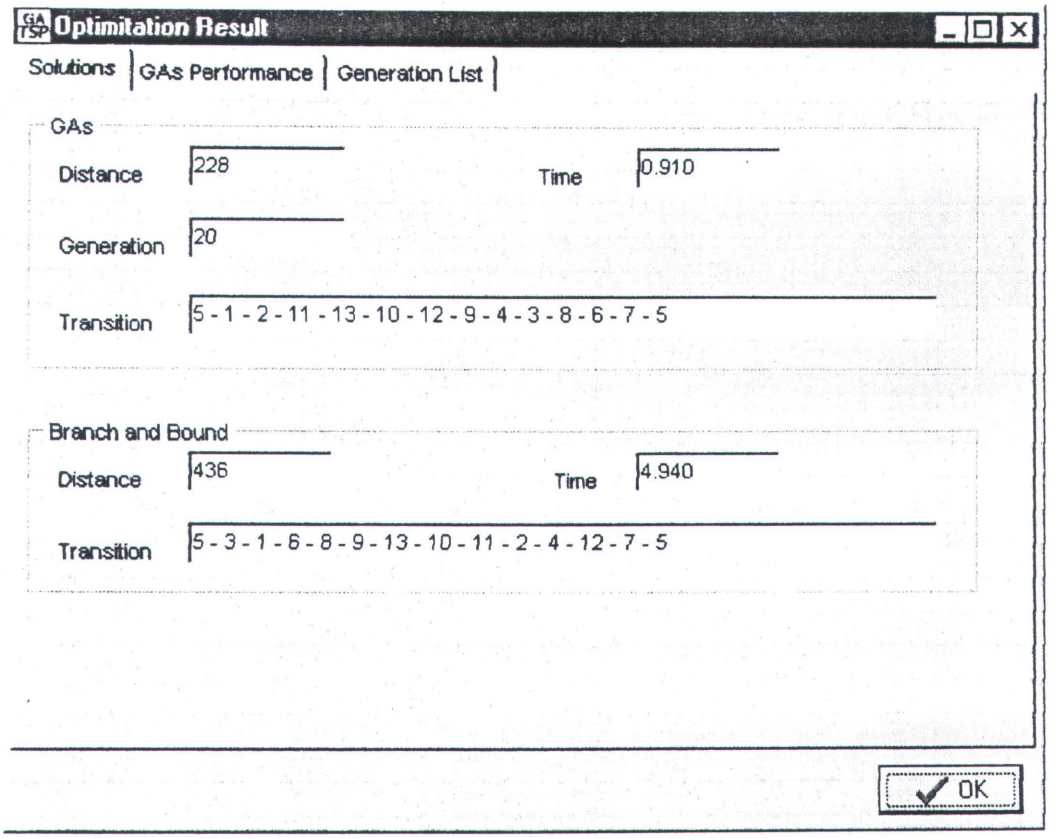
Selain itu, untuk jumlah kota yang besar (di atas 15 kota), metode Branch and Bound sangat boros dalam penggunaan memori (untuk pembentukan queue-nya). Dengan konfigurasi perangkat keras minimal seperti yang disebut di atas, Branch and Bound hanya bisa menyelesaikan permasalahan maksimal 15 kota. Sedangkan untuk Algoritma Genetika masih bisa untuk 100 kota.



Gambar 5.7 Hasil Uji Coba untuk 11 Kota

Dari sisi keoptimalan untuk penyelesaian walaupun tidak selalu Algoritma Genetika unggul, Algoritma Genetika relatif masih lebih unggul dibanding Branch and Bound. Artinya kalaupun Algoritma Genetika tidak lebih unggul dalam suatu waktu, hal tersebut bisa diatasi dengan meningkatkan unjuk kerja Algoritma Genetika dengan memilih parameter-parameter yang tepat untuk permasalahan yang diuji-cobakan. Hal ini disebabkan Branch and Bound merupakan metode yang *single source* dan dia mencari lintasan yang penuh untuk isi queue yang pertama. Artinya metode ini sangat bergantung dari mana dia memulai lintasan dan dengan mendapatkan lintasan yang penuh pertama kali belum tentu

merupakan lintasan yang optimum. Sedangkan Algoritma Genetika dengan sifatnya *mutiple source* karena mengekplorasi wilayah pencarian dan mengeksploitasi solusi terbaik, maka untuk permasalahan TSP, yang memang bentuk lintasannya siklik, dia tidak terpengaruh dari mana lintasan dimulai.



Gambar 5.8 Hasil Uji Coba untuk 13 Kota

BAB VI
PENUTUP

BAB VI

PENUTUP

Dalam bab ini diuraikan beberapa hal yang menjadi kesimpulan dari hasil uji coba perangkat lunak, serta beberapa saran bagi pembaca yang diharapkan di masa mendatang dapat menyempurnakan atau mengembangkan perangkat lunak ini dari sisi konsep maupun teknis, sehingga semakin berguna dalam kehidupan.

6.1 KESIMPULAN

Beberapa kesimpulan yang dapat ditarik dari hasil uji coba perangkat lunak ini adalah sebagai berikut:

1. Unjuk kerja Algoritma Genetika untuk penyelesaian TSP sangat dipengaruhi oleh parameter-parameter yang digunakan dan nilai-nilai dari parameter yang digunakan. Pengaruh parameter-parameter adalah sebagai berikut:
   - Untuk jumlah kota yang kecil, maka daerah pencarian cukup dengan ukuran populasi yang kecil. Namun untuk jumlah kota yang besar, maka ukuran populasi yang kecil sangat berpengaruh terhadap ruang pencarian. Artinya nilai optimum sulit didapatkan dalam kondisi seperti ini. Dalam Algoritma Genetika hal ini bisa diatasi dengan menambah ukuran populasi.
   - Normalisasi Linier merupakan mekanisme penyesuaian yang paling sesuai untuk menghindari terjadinya dominasi individu-individu tertentu pada ruang pencarian pada TSP.
   - Untuk menghindari konvergensi dini, maka harus dipilih metode mutasi yang tepat untuk permasalahan yang diselesaikan.

- Sedangkan metode seleksi campuran (Elitism + Roulette Wheel) sebagai metode seleksi yang paling tepat untuk penyelesaian TSP.

2. Jika Algoritma Genetika dioptimalkan, artinya pemilihan parameter dan nilai dari parameter diberikan tepat untuk eksplorasi ruang pencarian dan eksploitasi nilai optimum, maka Algoritma Genetika akan memberikan unjuk kerja yang lebih bagus dalam hal mendapatkan nilai optimum yang diinginkan. Hal ini dapat dibuktikan dengan uji coba yang hasilnya telah dijelaskan dalam bab sebelumnya.

6.2 SARAN UNTUK PENGEMBANGAN LEBIH LANJUT

Dengan adanya berbagai kelebihan dan kekurangan yang ada dalam perangkat lunak ini, berikut ini diberikan beberapa saran:

1 Untuk lebih meningkatkan unjuk kerja Algoritma Genetika dalam penyelesaian TSP ini, disarankan untuk melakukan uji coba yang intensif dengan inovasi guna pemilihan kombinasi parameter yang lebih sesuai.

2 Mengenai pengembangan dari sisi teknis perangkat lunak ini dapat diterapkan pada permasalahan riil yang menggunakan TSP, seperti desain jaringan telepon dan integrasi sirkuit yang membutuhkan perhitungan jarak terpendek dari lintasan yang berbentuk siklik.

DAFTAR PUSTAKA

DAFTAR PUSTAKA

Artanto, Nico. *Aplikasi Hopfield Memory untuk Penyelesaian Traveling Salesman Problem.* Jurusan Teknik Komputer Fakultas Teknologi Industri Institut Teknologi Sepuluh Nopember, Surabaya, 1995

Beasley, David, David R. Bull, Ralph R. Martin. An Overview of Genetic Algorithms: Part 1, Fundamentals. *University Computing*, Vol.15(2), pp:58-69, 1993

Cantu, Marco. *Mastering Delphi.* SYBEX Inc., San Francisco,1995

Chan, Sophia. *Genetic Algorithms, Genetic Algorithm vs Traditional Methods.* White Paper, 1996

Chan, Sophia. *Genetic Algorithms, John Holland and The Invention of Genetic Algorithm.* White Paper, 1996

Chan, Sophia. *Genetic Algorithms, The Origin of Genetic Algorithm.* White Paper, 1996

Filho, Jose Riberio, Cesare Alippi, Philip Treleaven. *Genetic Algorithm Programming Environments.* University College London, White Paper to appear in the IEEE COMPUTER Journal

Gen, Mitsuo, Runwei Cheng. *Genetic Algorithms and Engineering Design.* John Wiley & Sons, Inc., New York, 1997

Goldberg, David E. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley, Reading, MA, New York, 1989

Koza, John R. *Genetic Programming.* Fifth printing, Massachussetts Institute of Technology, Massachussetts, 1996

Ladd, Scott Robert. *Genetic Algorithm in C++.* M & T Books, New York, 1996

Michalewicz, Zbigniew. *Genetic Algorithms + Data Structure = Evolution Programs.* Third Revised and Extended Edition, Springer-Verlag Berlin Heidelberg, New York, 1996

Winston, Patrick Henry. *Artificial Intelligence*. Second Edition, Addison-Wesley Publishing Company, Massachussetts, 1984

LAMPIRAN

LAMPIRAN

PETUNJUK PENGGUNAAN PERANGKAT LUNAK

L.1 ANTARMUKA PERANGKAT LUNAK

Perangkat Lunak ini memiliki tiga menu utama pada tampilan masukan, yaitu menu **File, Run** dan **Help**. Selain menu utama tampilan juga dilengkapi dengan tiga tombol cepat (*speedbutton*). Tombol cepat yang pertama digunakan untuk operasi **Open,** tombol cepat yang kedua digunakan untuk **Run,** dan tombol cepat yang terakhir digunakan untuk **Exit** (keluar) dari perangkat lunak.

Menu File terdiri atas dua submenu yang mana fungsi masing-masing submenu tersebut seperti dijelaskan berikut ini :

- Open, untuk membuka file *.txt
- Exit, untuk keluar dari program

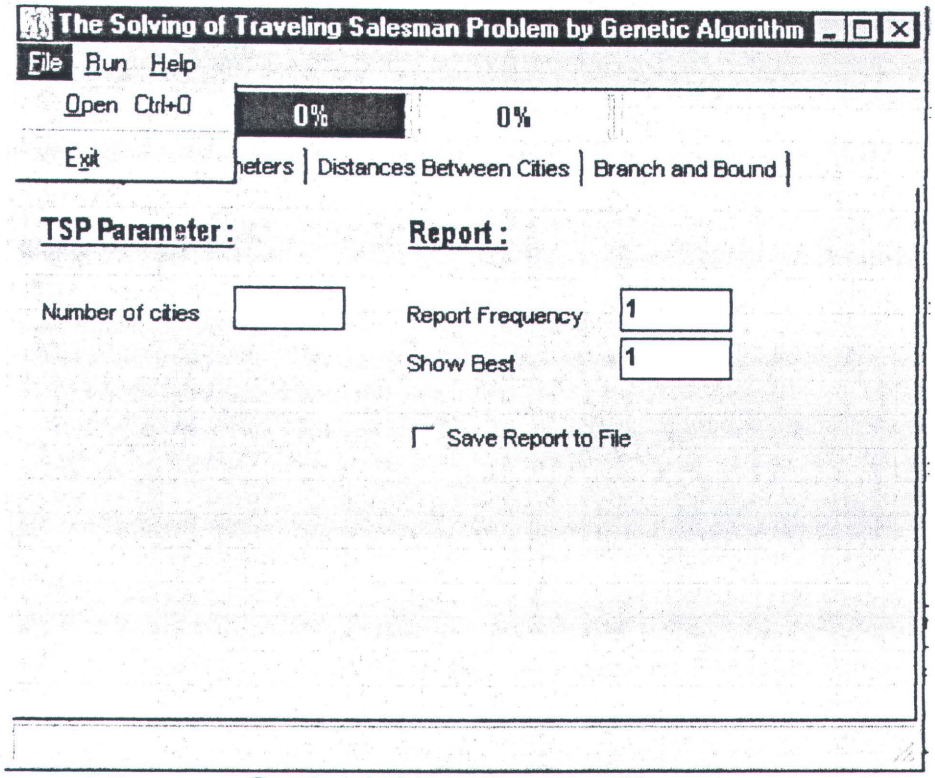
Tampilan dari Menu File ini diperlihatkan pada Gambar L.1.

Menu Run dengan submenu start berfungsi untuk menjalankan optimasi program. Tampilan menu Run ini dapat dilihat pada Gambar L.2.

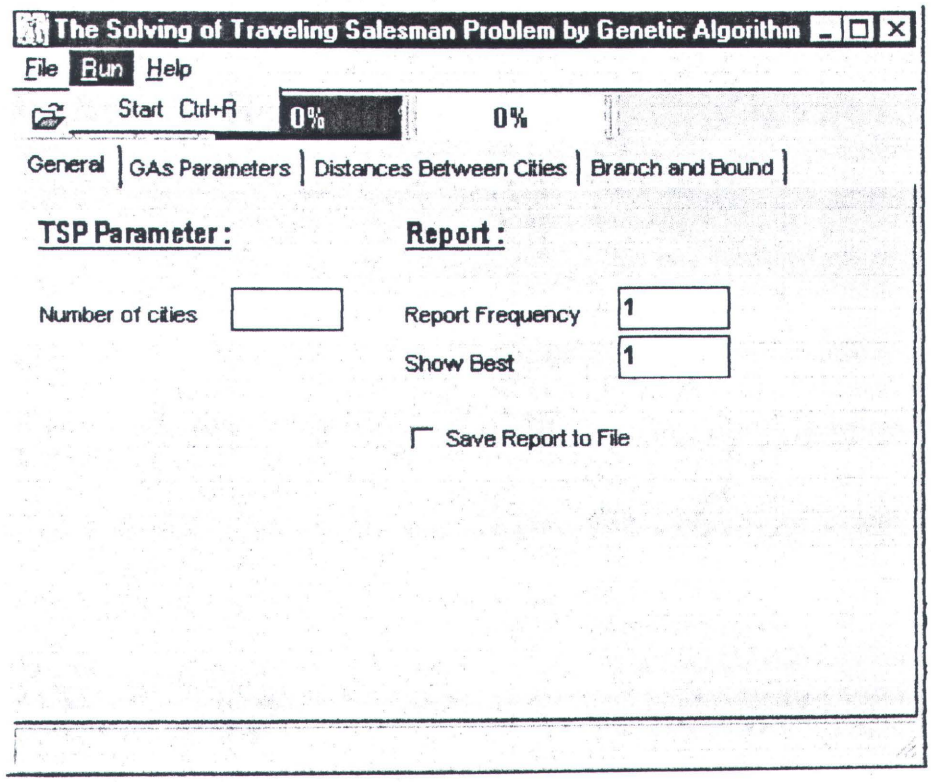
Menu Help berfungsi untuk memberikan informasi kepada pengguna. Menu ini terdiri atas dua submenu, yaitu:

- Parameter yang terdiri dari dua bagian, yaitu bagian General yang berisi keterangan mengenai fungsi parameter general yang ada. Bagian kedua adalah GAs yang berisi keterangan mengenai fungsi parameter Algoritma Genetika yang ada.
- About, yang berisi informasi mengenai perangkat lunak yang sedang aktif.

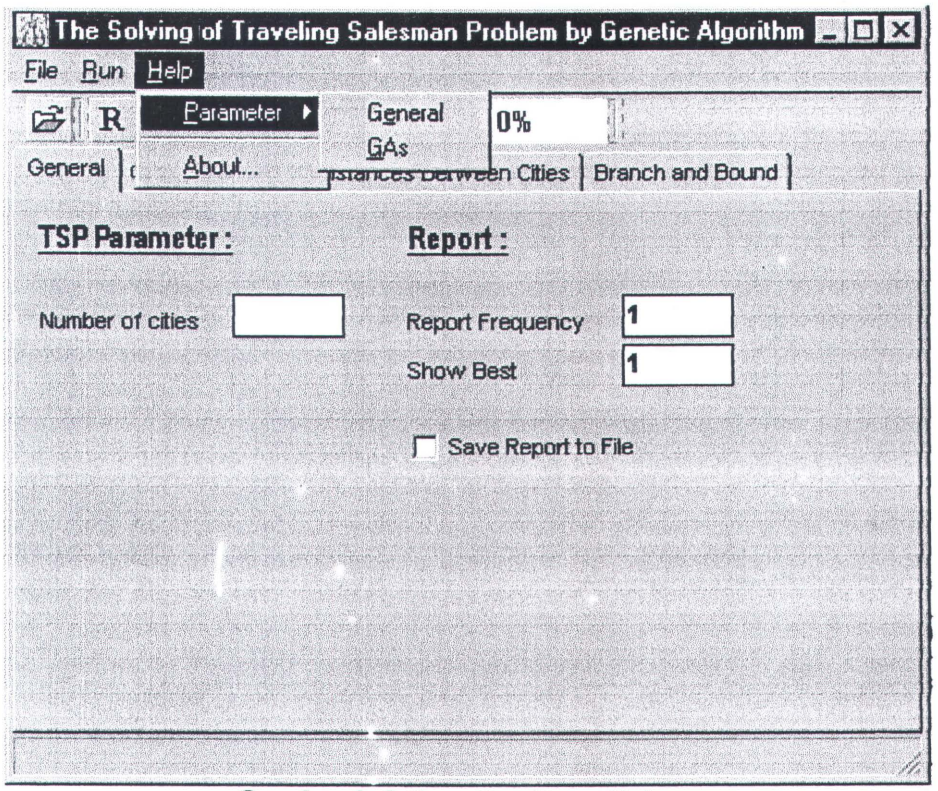
Tampilan menu help ini dapat dilihat pada Gambar L.3.



Gambar L.1 Tampilan Menu File



Gambar L.2 Tampilan Menu Run



Gambar L.3 Tampilan Menu Help

Adapun bagian tombol cepat yang ada, masing-masing untuk tombol pertama berfungsi sebagai tombol cepat untuk membuka file *.txt. Tombol kedua berfungsi untuk menjalankan optimasi. Dan tombol terakhir berfungsi untuk keluar dari aplikasi. Selain tiga tombol cepat, dalam panel tersebut terdapat dua buah *gauge* (obyek yang menunjukkan jalannya suatu proses). Gauge pertama adalah untuk menunjukkan jalannya proses Algoritma Genetika, dan gauge menunjukkan jalannya proses Branch and Bound.

Selain menu, button dan gauge yang dijelaskan di atas, dalam antar muka masukan juga terdapat 4 tabsheet. Masing-masing tabsheet tersebut dijelaskan di bawah ini.

Tabsheet pertama berisi General Parameter, dimana di dalamnya terdapat :

- Number of cities : berisi jumlah kota yang akan terisi dengan otomatis ketika file *.txt dibuka.

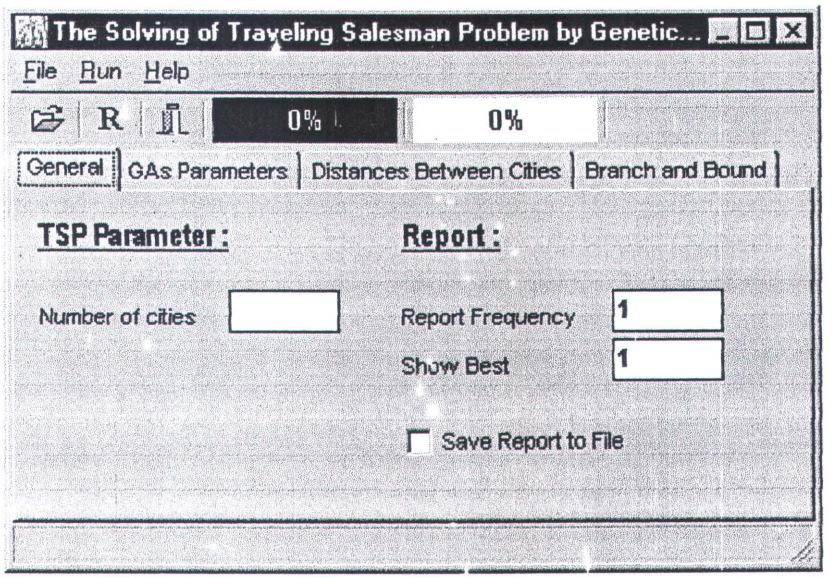
- Report Frequency : angka yang dibutuhkan untuk perangkat lunak melakukan pencatatan. Apabila bagian ini mencatat angka x maka untuk tiap kelipatan generasi x dilakukan pencatatan.
- Show Best : menunjukkan berapa populasi terbaik yang dicatat dari generasi ke-x (atau kelipatannya).

Tabsheet kedua adalah parameter Algoritma Genetika, yang mana fungsi dari masing-masing bagian adalah seperti telah dijelaskan dalam Bab IV. Hanya saja perlu dicatat bahwa groupbox yang berisi parameter Linear Normalitation hanya berfungsi jika Linier Normalisasi dipilih sebagai Mekanisme Penyesuaian.

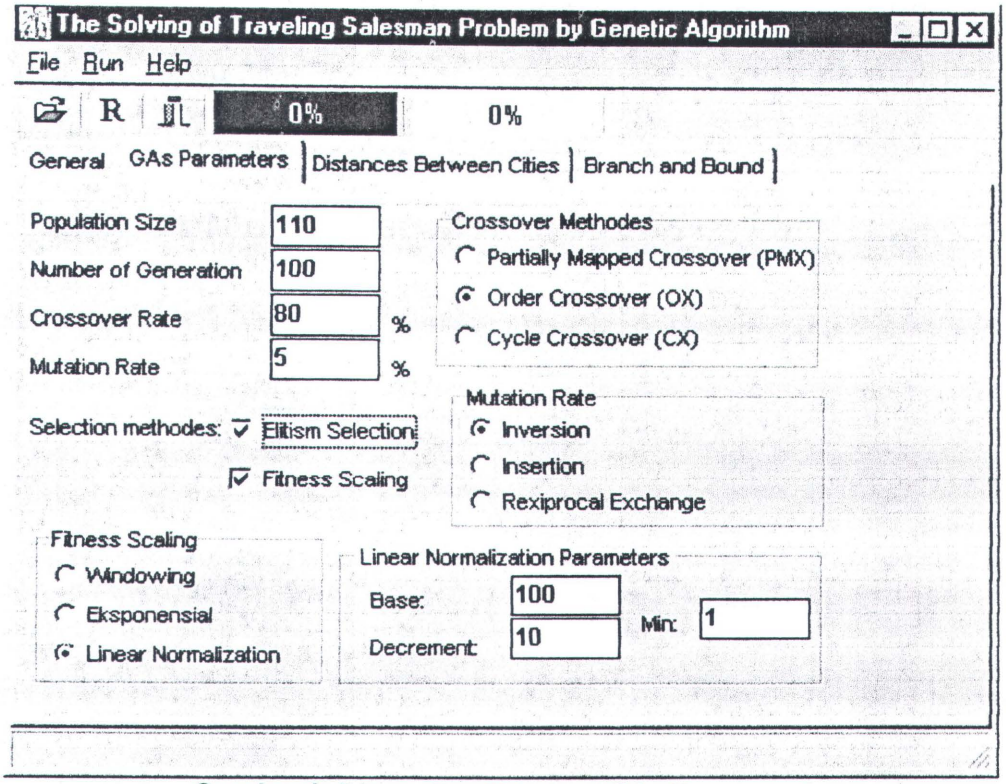
Tabsheet ketiga yaitu Distances Between Cities berisi mengenai tabel data jarak antar kota, yang akan terisi dengan otomatis jika file *.txt dibuka.

Tabsheet keempat yaitu Branch and Bound, di dalamnya hanya ada satu edit box, yang menunjukkan awal kota dimulainya lintasan TSP untuk penyelesaian metode Branch and Bound.

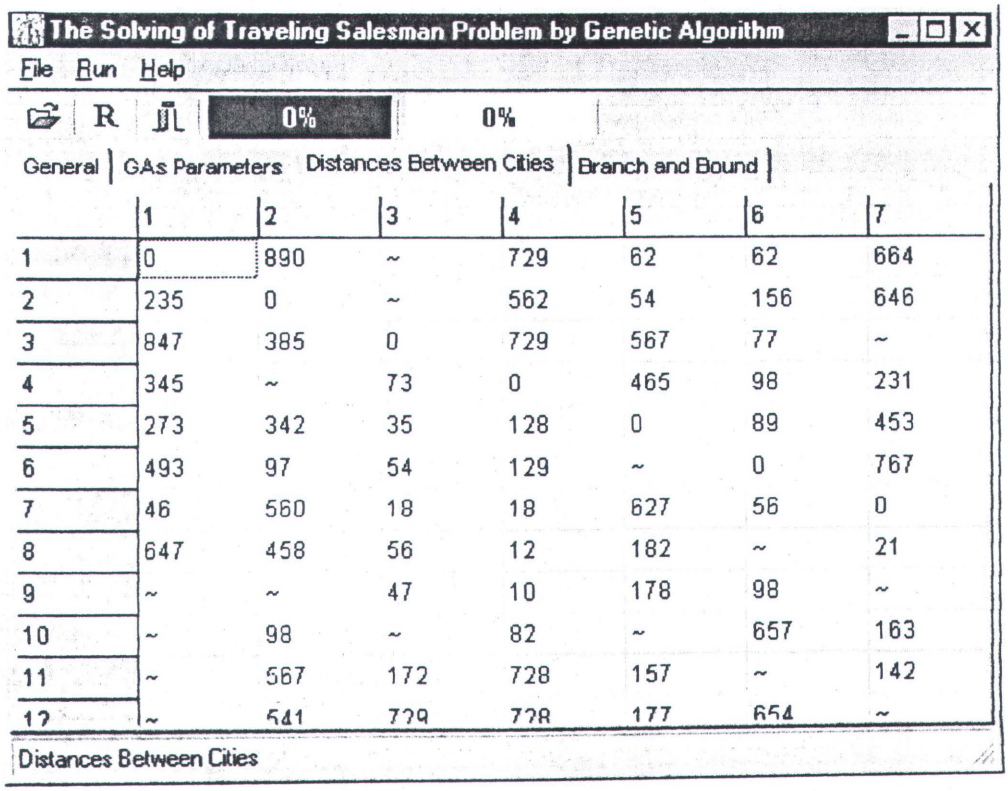
Tampilan masing-masing tabsheet ini diperlihatkan pada Gambar L.4 sampai dengan Gambar L.7.



Gambar L.4 Gambar Tabsheet General Parameter

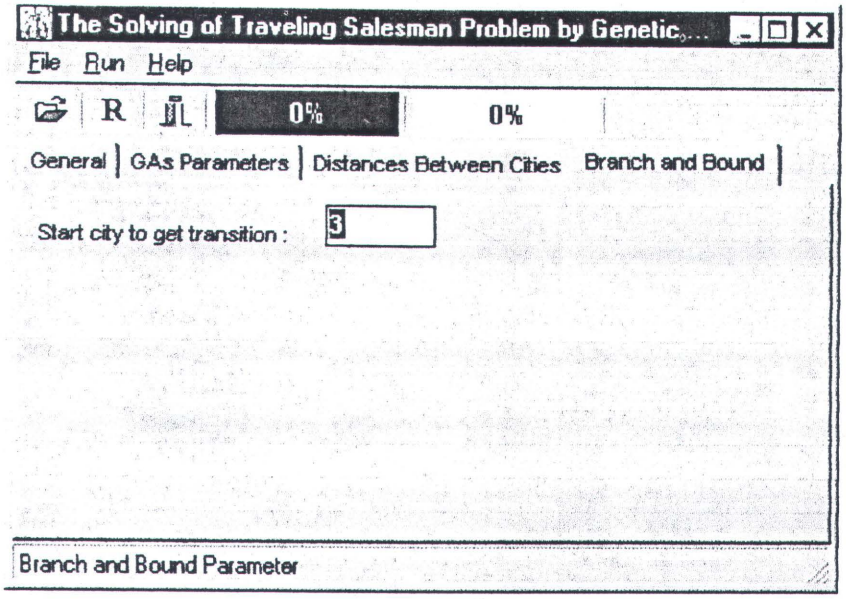


Gambar L.5 Tampilan Tabsheet GAs Parameter



	1	2	3	4	5	6	7
1	0	890	~	729	62	62	664
2	235	0	~	562	54	156	646
3	847	385	0	729	567	77	~
4	345	~	73	0	465	98	231
5	273	342	35	128	0	89	453
6	493	97	54	129	~	0	767
7	46	560	18	18	627	56	0
8	647	458	56	12	182	~	21
9	~	~	47	10	178	98	~
10	~	98	~	82	~	657	163
11	~	567	172	728	157	~	142
12	~	541	729	728	177	654	~

Gambar L.6 Tampilan Tabsheet Distances between Cities

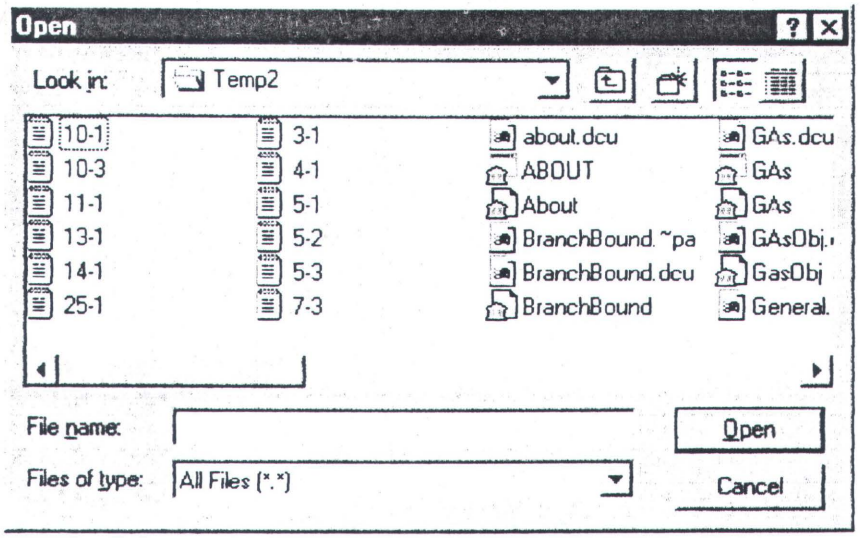


Gambar L.7 Tampilan Tabsheet Branch and Bound

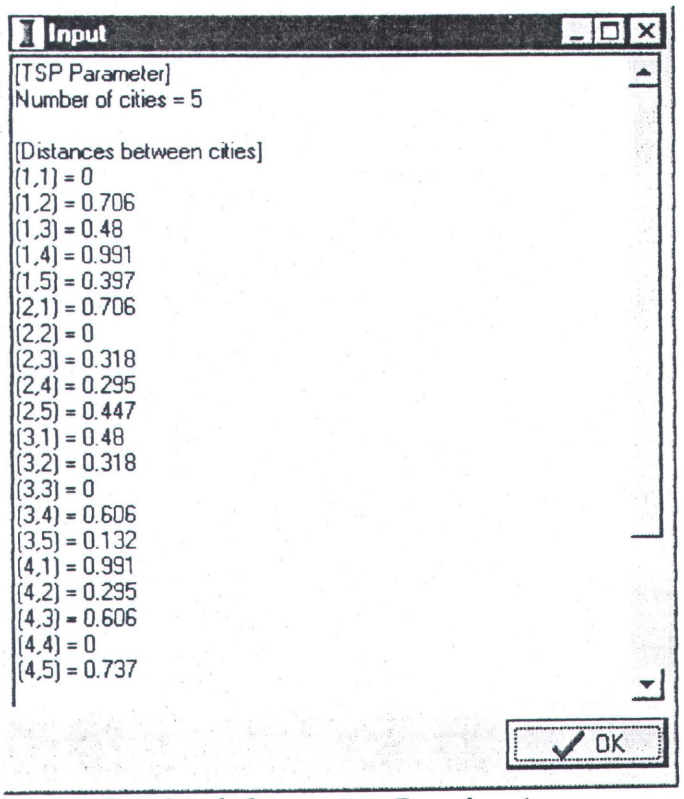
L.2 CARA PENGGUNAAN PERANGKAT LUNAK

Perangkat Lunak ini dapat digunakan apabila pengguna telah membuka file *.txt sebagai langkah awal. File ini dapat dibuka dengan menekan submenu **Open** pada menu **File** atau menekan speedbutton pertama. Setelah pilihan di atas ditekan akan muncul open dialog seperti pada Gambar L.8. Setelah file dibuka, maka informasi akan ditampilkan dalam sebuah form seperti dalam Gambar L.9. Setelah itu tekan tombol OK yang ada dalam form tersebut sehingga form tersebut ditutup, selanjutnya informasi yang ada dalam form tersebut oleh perangkat lunak ditransfer dalam antarmuka masukan

Dari sini dengan memasukkan parameter Algoritma Genetika baik di bagian tabsheet General ataupun GAs, program akan bisa dijalankan. Hasil running dari perangkat lunak ini akan ditampilkan dalam antarmuka keluaran.



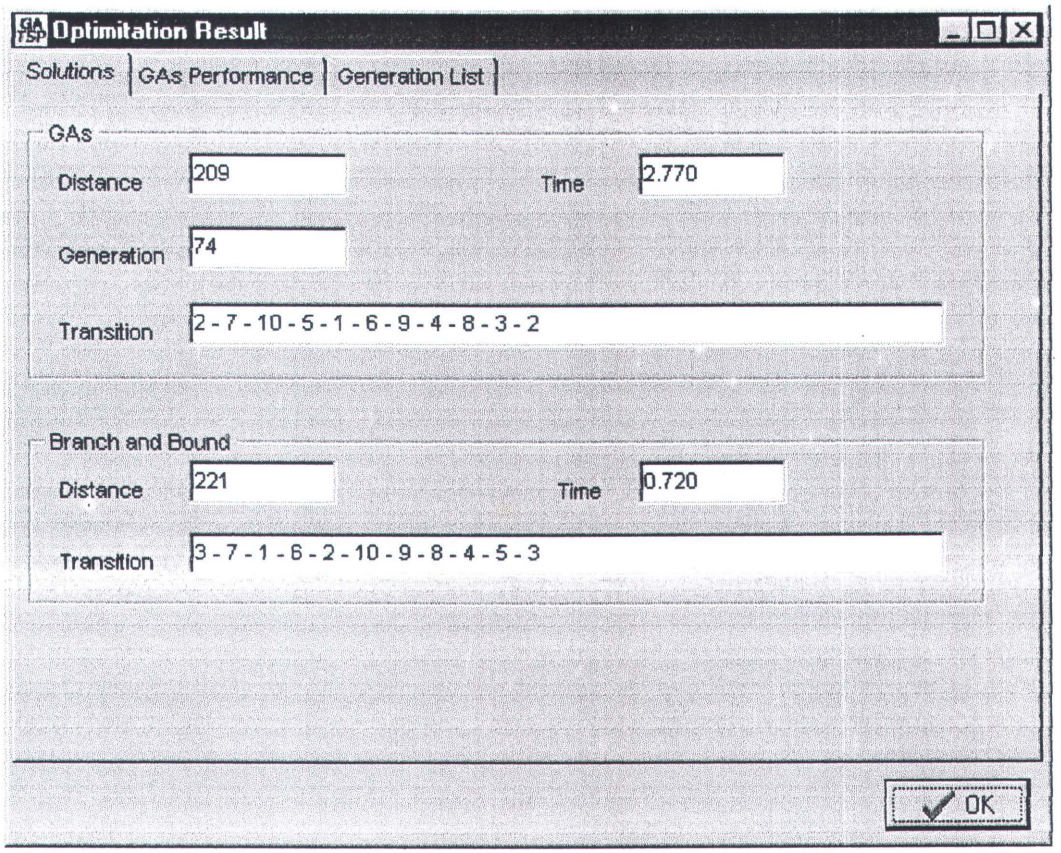
Gambar L.8 Tampilan Open Dialog



Gambar L.9 Tampilan Form Input

Dalam antarmuka keluaran, terdapat 3 tabsheet. Tabsheet pertama adalah untuk melihat hasil optimasi berupa lintasan terpendek dan jaraknya, yang diperoleh metode Algoritma Genetika maupun metode Branch and Bound.

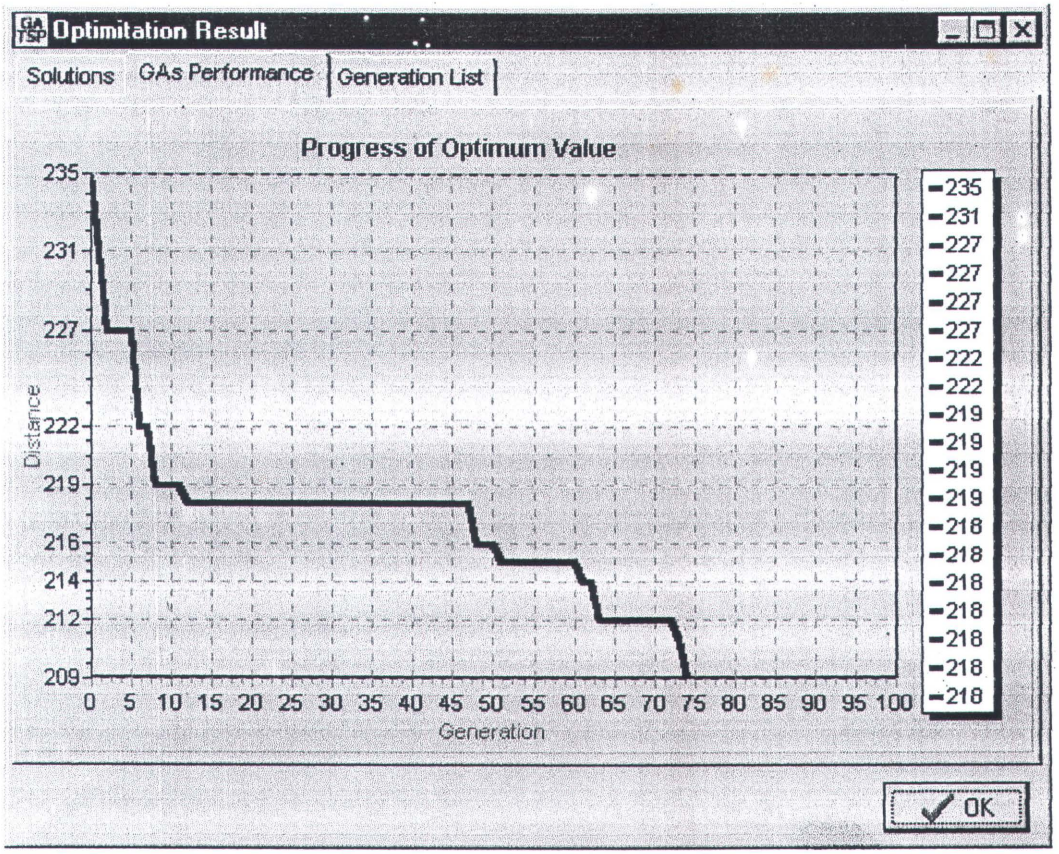
Tabsheet kedua berisi mengenai *report* berupa grafik perkembangan nilai optimum dari generasi ke generasi. Tabsheet ketiga berisi mengenai report perkembangan individu-individu terbaik dalam tiap generasi, yang ditunjukkan lintasannya dan jarak terpendek dalam tiap generasi. Tampilan dari tiap tabsheet ini dapat dilihat pada Gambar L.10 sampai dengan Gambar L.12.



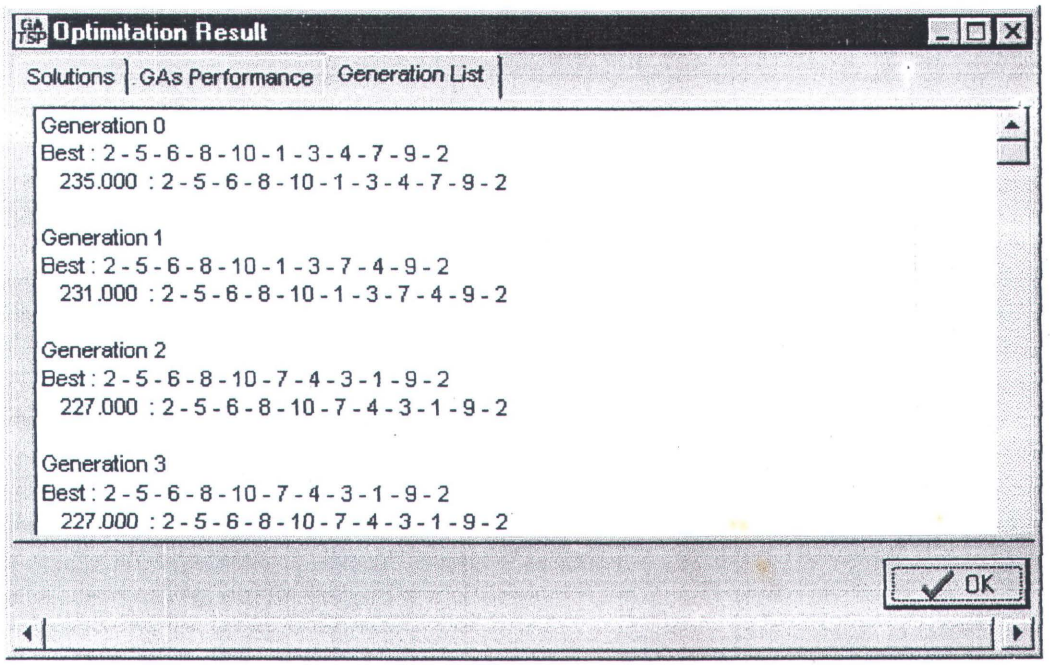
Gambar L.10 Tampilan Antarmuka Keluaran - Tabsheet Solutions

Antar muka keluaran akan muncul apabila kedua gauge pada panel yang terdapat pada antarmuka masukan telah menunjuk angka 100%.

Apabila pengguna ingin menjalankan proses yang lain, maka pengguna harus keluar dulu dari antar muka keluaran, dengan menekan tombol OK.



Gambar L.11 Tampilan Antarmuka Keluaran - Tabsheet GAs Performances



Gambar L.12 Tampilan Antarmuka Keluaran - Tabsheet Generation List