PENERAPAN ALGORITMA GENETIK PADA DNA

SEQUENCING BY HYBRIDIZATION

NOVI MURNIATI

0606067635

UNIVERSITAS INDONESIA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

DEPARTEMEN MATEMATIKA

DEPOK

2009

PENERAPAN ALGORITMA GENETIK PADA DNA

SEQUENCING BY HYBRIDIZATION

Skripsi ini diajukan sebagai salah satu syarat

untuk memperoleh gelar Sarjana Sains

Oleh :

NOVI MURNIATI

0606067635

DEPOK

2009

SKRIPSI : PENERAPAN ALGORITMA GENETIK PADA DNA *SEQUENCING BY HYBRIDIZATION*

NAMA : NOVI MURNIATI

NPM : 0606067635

SKRIPSI INI TELAH DIPERIKSA DAN DISETUJUI

DEPOK, 18 NOVEMBER 2009

Dra. Denny Riama Silaban, M.Kom.	Dhian Widya, S.Si., M.Kom.
PEMBIMBING I	PEMBIMBING II

Tanggal Lulus Ujian Sidang Sarjana: 17 Desember 2009

Penguji I : Dra. Denny Riama Silaban, M.Kom.

Penguji II : Dr. Al Haji Akbar B., M.Sc.

Penguji III : Prof. Dr. Djati Kerami

KATA PENGANTAR

Puji syukur kepada Allah SWT atas segala berkah dan karunia-Nya sehingga penulisan skripsi ini selesai tepat pada waktunya.

Penulisan skripsi ini selesai atas dukungan dan bantuan dari berbagai pihak. Oleh karena itu, pada kesempatan ini penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Ibu dan Bapak selaku orang tua yang selalu memberikan doa, semangat, dan dukungan bagi penulis.
2. Dra. Denny Riama Silaban, M.Kom. dan Dhian Widya, S.Si., M.Kom. selaku pembimbing skripsi yang telah banyak meluangkan waktu dan pikiran serta memberikan masukan-masukan untuk penulis dalam menyelesaikan skripsi ini.
3. Bevina D. Handari,Phd., Dra. Siti Aminah, M.Kom., Dr. Kiki Ariyanti S., dan Prof. Dr. Djati Kerami yang telah hadir dan memberikan saran-saran bagi penulis pada SIG 1, SIG 2, dan kolokium.
4. Seluruh dosen dan staff Departemen Matematika FMIPA UI, khususnya Dra. Nora Hariadi, M.Sc. selaku pembimbing akademik penulis selama menjalani masa kuliah.
5. Mas Di dan Mas Per selaku kakak yang juga telah memberikan semangat dan dukungan kepada penulis terutama selama penyusunan skripsi ini.

6. Budi Utami, Widya, dan Noor Indah Ekawati selaku teman yang telah banyak membantu dalam melakukan simulasi program dan *editing* skripsi.
7. Farah Amalia dan Herlina. Terima kasih atas bantuan penginapan saat penyusunan skripsi dan saat menjelang pelaksanaan SIG 1, SIG 2, dan kolokium.
8. Inne, Mei, Anggha, Stefani, Rahanti, dan Ar-Rizqiyatul yang telah berjuang bersama selama penyusunan skripsi.
9. Alfa, Puspa, Arumella, Yunita, Lena, Milla, Widya, dan anak kumpulan "bekel ++" lainnya. Terimakasih atas persahabatan yang telah terjalin selama ini dan semoga sampai seterusnya.
10. Teman-teman Matematika angkatan 2006. Tetap semangat ya kalian semua.

Penulis juga ingin mengucapkan terima kasih kepada seluruh pihak yang tidak dapat disebutkan satu per satu, yang telah membantu dalam penyusunan skripsi ini. Akhir kata, penulis mohon maaf jika terdapat kesalahan atau kekurangan dalam skripsi ini. Penulis berharap semoga skripsi ini bermanfaat bagi pembaca.

Penulis

2009

ABSTRAK

DNA *Sequencing by Hybridization* (DNA SBH) adalah suatu proses pembentukan barisan nukleotida suatu rantai DNA dari kumpulan fragmen yang disebut spektrum. Spektrum tersebut diperoleh dari proses biokimia yang disebut hibridisasi. DNA SBH dapat dipandang sebagai masalah optimisasi yang dapat diselesaikan dengan menggunakan algoritma genetik. Prinsip kerja algoritma genetik berdasarkan pada teori evolusi Charles Darwin. Pada skripsi ini akan dibahas penerapan kinerja algoritma genetik pada DNA SBH. Terdapat tiga tahapan penting dalam algoritma genetik, yakni proses seleksi, *crossover*, dan mutasi. Jenis metode yang digunakan pada proses seleksi, *crossover*, dan mutasi secara berturut-turut adalah metode yang merupakan kombinasi antara *roulette wheel* dan *deterministic*, *structured crossover*, dan *swap mutation*. Kinerja algoritma genetik akan diuji dengan menggunakan data dari *Gen Bank* dan masalah DNA SBH yang dibuat secara acak. Selain itu juga akan dilihat pengaruh perubahan nilai probabilitas *crossover* (*c*) dan probabilitas mutasi (*m*) terhadap kinerja algoritma genetik untuk DNA SBH. Berdasarkan hasil percobaan diperoleh bahwa algoritma genetik cukup baik digunakan pada DNA SBH. Selain itu, perubahan nilai probabilitas *crossover* (*c*) dan probabilitas mutasi (*m*) ternyata mempengaruhi kinerja algoritma genetik dalam memperoleh solusi.

Kata kunci: algoritma genetik; DNA *Sequencing by Hybridization*
xi+94 hlm.; lamp.
Bibilografi: 10 (2004-2009)

DAFTAR ISI

Halaman

DAFTAR TABEL

DAFTAR GAMBAR

LAMPIRAN

Halaman

BAB I

PENDAHULUAN

1.1 LATAR BELAKANG MASALAH

DNA (*Deoxyribose Nucleic Acid*) adalah asam nukleat yang mengandung materi genetik dan berfungsi untuk mengatur perkembangan biologis seluruh bentuk kehidupan secara seluler [Wik 09]. DNA memiliki struktur *double helix* yang antiparalel dengan komponen-komponennya, yaitu gula pentosa (deoksiribosa), gugus fosfat, dan pasangan basa. Pasangan basa pada DNA terdiri atas dua macam, yaitu basa purin dan pirimidin. Basa purin terdiri atas adenin (*A*) dan guanin (*G*) yang memiliki struktur cincin-ganda, sedangkan basa pirimidin terdiri atas sitosin (*C*) dan timin (*T*) yang memiliki struktur cincin-tunggal [Wik 09]. Ketika guanin berikatan dengan sitosin, maka akan terbentuk tiga ikatan hidrogen, sedangkan ketika adenin berikatan dengan timin maka hanya akan terbentuk dua ikatan hidrogen. Satu komponen pembangun DNA disebut nukleotida. Nukleotida terdiri atas satu gula pentosa, satu gugus fosfat, dan satu basa nitrogen. DNA dapat memiliki jutaan nukleotida yang terangkai seperti rantai, misalnya DNA manusia yang terdiri atas 220 juta nukleotida [DS 09]. Dengan kata lain, barisan DNA

merupakan barisan nukleotida yang membentuk DNA. Karena tiap nukleotida hanya mengandung satu jenis basa nitrogen (adenin, sitosin, timin, atau guanin) maka suatu barisan DNA biasanya dinyatakan sebagai barisan dari alfabet *A*, *C*, *T*, dan *G* yang merepresentasikan nukleotida yang membentuk rantai DNA tersebut. Barisan nukleotida yang pendek dinamakan oligonukleotida atau fragmen.

Barisan nukleotida pada DNA menyimpan kode genetik. Barisan nukleotida yang berbeda akan menyimpan kode genetik yang berbeda pula. Barisan DNA pada setiap organisme bersifat unik, hal ini berarti antara satu organisme dengan organisme lainnya dapat dibedakan dari struktur barisan DNA-nya. Oleh karena itu, untuk mengidentifikasi suatu organisme dapat dilakukan dengan cara melihat struktur barisan DNA organisme tersebut. Namun demikian, seperti yang dijelaskan sebelumnya bahwa suatu rantai DNA dapat memiliki jutaan nukleotida, tetapi terkadang dalam satu kali eksperimen hanya barisan yang terdiri dari sekitar 500-800 nukleotida yang mampu dibaca [DS 09]. Oleh karena itu, untuk membaca barisan nukleotida pada rantai DNA yang panjang, rantai DNA dibagi menjadi beberapa fragmen-fragmen yang lebih pendek. Kemudian dari fragmen-fragmen tersebut akan dicari barisan DNA asal dengan memperhatikan tumpang tindih (*overlap*) antar fragmen. Proses pembentukan barisan DNA tersebut dinamakan dengan DNA *sequencing*. Jadi yang dimaksud dengan DNA

sequencing adalah proses pembentukan barisan nukleotida dari suatu rantai DNA.

Salah satu metode DNA *sequencing* adalah *Sequencing by Hybridization* (SBH). SBH merupakan suatu proses pembentukan barisan nukleotida suatu rantai DNA dari kumpulan fragmen yang disebut spektrum. Spektrum tersebut diperoleh dari proses biokimia yang disebut hibridisasi [BK 06]. Hibridisasi yang berlangsung kadangkala menghasilkan suatu *error* negatif ataupun *error* positif pada spektrum yang terbentuk. *Error* positif terjadi jika terdapat fragmen pada spektrum yang bukan merupakan bagian dari barisan DNA asal. Sedangkan *error* negatif terjadi jika terdapat fragmen yang merupakan bagian dari barisan DNA asal namun fragmen tersebut tidak terdapat pada spektrum. *Error* pada spektrum berpengaruh pada proses DNA *sequencing*, karena akan menyebabkan barisan DNA yang terbentuk berbeda dari barisan DNA asal. Spektrum yang tidak mengandung *error* disebut spektrum ideal.

SBH dapat dilakukan dengan berbagai cara, antara lain dengan menggunakan *de Bruijn graphs* dan *Eulerian circuit.* SBH juga bisa dipandang sebagai masalah optimisasi. Pada skripsi ini, SBH akan dipandang sebagai masalah optimisasi kombinatorik, yakni menentukan kombinasi dari urutan fragmen anggota spektrum yang membentuk suatu barisan DNA dengan panjang tertentu sedemikian sehingga *error* positif dan *error* negatif dari pembentukan barisan DNA minimum. Masalah optimisasi

kombinatorik tersebut akan diselesaikan dengan menggunakan algoritma genetik.

Algoritma genetik merupakan salah satu algoritma yang biasa digunakan untuk menyelesaikan masalah optimisasi kombinatorik. Algoritma genetik pertama kali diperkenalkan oleh John Holland sekitar tahun 1970-an [IGA 09]. Prinsip kerja dari algoritma genetik didasari oleh teori evolusi Charles Darwin [MA 09]. Secara umum, tahapan dari algoritma genetik diawali dengan pembentukan populasi awal. Selanjutnya dilakukan proses-proses yang menggunakan operator genetik seleksi, *crossover*, dan mutasi untuk membentuk populasi baru yang akan digunakan untuk generasi berikutnya. Banyaknya proses *crossover* dan mutasi yang terjadi bergantung pada nilai parameter probabilitas *crossover* dan probabilitas mutasi. Proses atau tahapan pada algoritma genetik tersebut dilakukan berulang kali sampai mencapai suatu kriteria berhenti yang ditentukan. Kriteria berhenti yang digunakan pada skripsi ini adalah batas generasi. Jadi, algoritma genetik akan berhenti setelah mencapai batas generasi yang telah ditentukan.

1.2 PERUMUSAN MASALAH

Apakah algoritma genetik cukup baik digunakan dalam melakukan DNA *sequencing*? Apakah perubahan parameter probabilitas *crossover* dan

probabilitas mutasi mempengaruhi kinerja dari algoritma genetik dalam memperoleh solusi?

1.3 TUJUAN PENULISAN

Berdasarkan permasalahan yang dirumuskan sebelumnya, pembahasan skripsi ini bertujuan untuk melihat keakuratan algoritma genetik dalam melakukan DNA *sequencing*. Keakuratan diukur dari kedekatan solusi yang diperoleh dengan DNA *sequence* asli dari masalah DNA *sequencing* yang diambil dari *Gen Bank* dan beberapa masalah DNA SBH yang dibuat secara acak. Selain itu, akan dilihat pengaruh perubahan parameter probabilitas *crossover* dan probabilitas mutasi terhadap kinerja algoritma genetik untuk DNA *sequencing*.

1.4 PEMBATASAN PEMBAHASAN

Algoritma genetik yang digunakan untuk DNA *sequencing* dalam skripsi ini adalah algoritma genetik yang menggunakan kombinasi antara metode *roulette wheel* dan *deterministic* pada proses seleksi, metode *structured crossover* pada proses *crossover*, dan metode *swap mutation* pada proses mutasi.

1.5 SISTEMATIKA PENULISAN

Pembahasan selanjutnya dilakukan dengan sistematika sebagai berikut: Bab II berisi landasan teori yang membahas mengenai DNA *sequencing by hybridization* (DNA SBH) dan algoritma genetik. Bab III berisi pembahasan mengenai penerapan algoritma genetik dalam DNA *sequencing by hybridization* (DNA SBH). Bab IV berisi implementasi algoritma genetik untuk DNA *sequencing by hybridization* (DNA SBH) dan beberapa hasil percobaan beserta analisisnya. Bab V berisi kesimpulan.

BAB II

LANDASAN TEORI

2.1 DNA *SEQUENCING BY HYBRIDIZATION* (DNA SBH)

Hibridisasi merupakan suatu proses biokimia, dimana rantai tunggal DNA diurai menjadi kumpulan fragmen atau oligonukleotida. Kumpulan fragmen tersebut dinamakan spektrum yang biasa dinotasikan dengan *S*. Fragmen adalah barisan nukleotida yang pendek. Sedangkan nukleotida merupakan satu komponen pembangun DNA yang terdiri atas 1 gula pentosa, 1 gugus fosfat, dan 1 basa nitrogen (adenin, sitosin, timin, atau guanin). Karena pada tiap nukleotida hanya terdapat satu jenis basa nitrogen, maka suatu nukleotida biasanya dinyatakan dengan *A* (adenin), *C* (sitosin), *T* (timin), atau *G* (guanin) sesuai jenis basa nitrogen yang terkandung dalam nukleotida. Hibridisasi secara klasik biasanya menggunakan prinsip *isometric oligonucleotide library of size* untuk mendapatkan spektrum [JGKM 06]. Jadi, spektrum yang terbentuk dari hibridisasi ini terdiri dari kumpulan fragmen dengan panjang sama. Panjang suatu fragmen didefinisikan sebagai banyaknya nukleotida pada fragmen tersebut.

Contoh 2.1

Diberikan *S* = {*AAC, ACG, ATA, ATT, CGT, GTA, TAA, TAT*}. Spektrum *S* terdiri dari fragmen *AAC, ACG, ATA, ATT, CGT, GTA, TAA*, dan *TAT*. Tiap fragmen anggota spektrum dibentuk oleh 3 nukleotida, sehingga tiap fragmen dikatakan memiliki panjang 3. Jadi, spektrum *S* merupakan salah satu contoh spektrum hasil dari hibridisasi secara klasik yang menggunakan *isometric oligonucleotide library* dengan panjang 3.

Selain menggunakan prinsip *isometric oligonucleotide library*, hibridisasi dapat juga dilakukan dengan menggunakan prinsip *isothermic oligonucleotide library,* yakni tiap fragmen anggota spektrum hasil hibridisasi memiliki temperatur yang sama. Suatu *isothermic oligonucleotide library L* dengan temperatur t_L adalah sekumpulan fragmen yang memenuhi hubungan atau relasi

$$w_A x_A + w_T x_T + w_C x_C + w_G x_G = t_L, w_A = w_T, w_C = w_G, \text{dan } 2w_A = w_C$$

dengan w_i menyatakan *melting temperature* dari nukleotida jenis *i*, sedangkan x_i menyatakan banyaknya kemunculan nukleotida jenis *i* pada fragmen, dimana $i \in A,T,C,G$. Dalam hal ini $w_A = w_T = 2$ dan $w_C = w_G = 4$ [BK 06]. Jumlah *melting temperature* tiap nukleotida pada fragmen didefinisikan sebagai temperatur (*t*) dari fragmen.

Contoh 2.2

Contoh *isothermic oligonucleotide library*. Diberikan*S* = {*AAC, GTA, ATAA, CG*}. Spektrum *S* terdiri dari fragmen *AAC, GTA, ATAA,* dan *CG*. Temperatur dari tiap fragmen tersebut adalah

$$t(AAC) = (2 \times w_A) + w_C = (2 \times 2) + 4 = 8$$

$$t(GTA) = w_G + w_T + w_A = 4 + 2 + 2 = 8$$

$$t(ATAA) = (3 \times w_A) + w_T = (3 \times 2) + 2 = 8$$

$$t(CG) = w_C + w_G = 4 + 4 = 8$$

Karena temperatur tiap fragmen pada spektrum *S* bernilai 8, maka spektrum tersebut merupakan contoh spektrum hasil dari hibridisasi dengan *isothermic oligonucleotide library* dengan temperatur 8.

DNA *sequencing* adalah proses pembentukan barisan nukleotida dari suatu rantai DNA. Jadi, yang dimaksud dengan *DNA sequencing by hybridization* (DNA SBH) adalah proses pembentukan barisan nukleotida suatu rantai DNA dari spektrum yang dihasilkan dari hibridisasi [BK 06]. SBH diperkenalkan oleh Southern (1988), Bains dan Smith (1988), dan Drmanac (1989) [JGKM 06]. DNA SBH dibagi menjadi dua tahap. Tahap pertama adalah proses biokimia yang akan menghasilkan suatu spektrum. Tahap selanjutnya adalah merekonstruksi sebuah barisan dari spektrum tersebut [JGKM 06].

Pembahasan dalam skripsi ini difokuskan pada pembentukan barisan DNA yang sebenarnya sudah diketahui barisan nukleotidanya. Jadi, suatu rantai DNA yang sudah diketahui barisan nukleotidanya dihibridisasi hingga didapatkan suatu spektrum. Dari spektrum tersebut akan dibentuk kembali barisan DNA dengan menggunakan algoritma genetik.

Suatu hibridisasi tidak selalu berlangsung dengan sempurna, kadang menghasilkan suatu *error*. *Error* yang terjadi pada proses hibridisasi bisa berupa *error* positif atau *error* negatif [JGKM 06]. *Error* positif terjadi jika terdapat suatu fragmen di dalam spektrum yang bukan merupakan bagian dari rantai DNA asal. Sedangkan *error* negatif terjadi jika terdapat fragmen yang sebenarnya merupakan bagian dari barisan DNA asal tetapi tidak terdapat di dalam spektrum. *Error* negatif biasanya terjadi karena dalam hibridisasi, dua fragmen identik yang merupakan bagian dari rantai DNA asal hanya muncul satu kali dalam spektrum. Terjadinya *error* dalam hibridisasi mengakibatkan barisan DNA yang terbentuk dari spektrum bukanlah barisan DNA yang dimaksud. Untuk lebih jelas dalam memahami perbedaan antara *error* positif dan *error* negatif diberikan Contoh 2.3.

Contoh 2.3

Misalkan diberikan barisan DNA : *ATAACGTATA.* Spektrum yang dihasilkan : {*AAC, ACG, ACT, ATA, CGT, GTA, TAA, TAT*}. Pada spektrum terdapat fragmen *ACT*, sedangkan pada barisan DNA asal, fragmen *ACT* tidak muncul. Munculnya fragmen *ACT* pada spektrum inilah yang disebut sebagai

error positif. Pada barisan DNA asal terlihat bahwa fragmen *ATA* muncul dua kali. Namun di dalam spektrum fragmen *ATA* muncul hanya sekali. Hal inilah yang dinamakan *error* negatif.

DNA sequencing dapat dilakukan dengan menggunakan metode yang dipelajari dalam teori graf yakni *de Bruijn graphs* dan *Eulerian circuit .* Selain itu, masalah *DNA sequencing* juga dapat dipandang sebagai masalah optimisasi. Masalah optimisasi pada DNA SBH dipandang sebagai masalah optimisasi kombinatorik sebagaimana yang terjadi pada *Travelling Salesman Problem.* Masalah optimisasi kombinatorik yang dimaksud adalah menentukan kombinasi dari urutan fragmen anggota spektrum yang membentuk suatu barisan DNA dengan panjang tertentu sedemikian sehingga *error* positif dan *error* negatif dari pembentukan barisan DNA tersebut minimum. Masalah optimisasi tersebut dapat diselesaikan dengan menggunakan algoritma genetik.

2.2 ALGORITMA GENETIK

Algoritma genetik merupakan suatu jenis algoritma *metaheuristic*, tepatnya jenis *evolutionary metaheuristic* yang dapat digunakan untuk menyelesaikan masalah optimisasi [MA 09]. Algoritma genetik pertama kali diperkenalkan sekitar tahun 1970-an oleh John Holland dari *Michigan University* [IGA 09]. Prinsip kerja dari algoritma genetik menggunakan teknik-

teknik yang terinspirasi dari teori evolusi biologi seperti penurunan sifat, mutasi, seleksi, dan *crossover* (persilangan).

Dalam algoritma genetik, solusi dari suatu masalah direpresentasikan sebagai suatu individu. Suatu individu kadang direpresentasikan sebagai kumpulan gen yang disebut kromosom atau *genome*. Dengan menggunakan algoritma genetik, solusi yang dihasilkan belum tentu merupakan solusi eksak dari masalah optimisasi yang diselesaikan. Namun demikian, solusi yang didapat cukup mendekati solusi eksak yang dimaksud. Algoritma genetik menjadi sangat penting ketika masalah optimisasi yang akan diselesaikan cukup sulit untuk diformulasikan ke dalam suatu bentuk pemrograman matematika atau masalah tersebut dapat diformulasikan tetapi cukup rumit untuk diselesaikan karena sifat masalah yang terlalu kompleks.

Dalam suatu masalah optimisasi, dikenal suatu istilah fungsi objektif atau fungsi tujuan yang merupakan fungsi yang ingin dioptimalkan. Dalam algoritma genetik, fungsi yang akan dioptimalkan tersebut dinamakan fungsi *fitness*.

Tahapan dari algoritma genetik yakni membentuk suatu populasi (sekumpulan individu) awal, mengevaluasi populasi tersebut dengan suatu fungsi *fitness*, dan selanjutnya menerapkan prinsip-prinsip evolusi dengan menggunakan operator-operator genetik seperti seleksi, *crossover,* dan mutasi untuk membentuk populasi baru. Proses ini diulang hingga mencapai kriteria berhenti. Keefektifan suatu algoritma genetik bergantung pada

bagaimana cara pendefinisian populasi awal, ukuran populasi, operator genetik yang meliputi operator seleksi, *crossover*, dan mutasi, fungsi *fitness*, serta kriteria berhenti. Berikut akan dijelaskan mengenai tahapan-tahapan dari algoritma genetik.

2.2.1 Pembentukan Populasi Awal

Langkah awal dari algoritma genetik adalah menentukan representasi kromosom untuk tiap individu dari populasi. Ada beberapa jenis representasi yang biasa digunakan, yakni: *binary representation*, *representation in multiple matters*, *floating representation*, dan *representation in the form of tree* [Wik 09]. Pada *binary representation*, kromosom direpresentasikan dengan untaian bit biner (0 dan 1). Pada *representation in multiple matters*, kromosom terdiri dari beberapa karakter berbeda (*character string*). Pada *floating representation*, kromosom direpresentasikan dengan *floating number*. Sedangkan pada *representation in the form of tree*, tiap individu merupakan simpul (*node*) dari suatu pohon (*tree)*. Jenis representasi yang digunakan tergantung pada masalah optimisasi yang akan diselesaikan.

Pada algoritma genetik yang dijelaskan pada skripsi ini, jenis representasi kromosom yang digunakan adalah *representation in multiple matters.* Langkah awalnya adalah mengkodekan fragmen-fragmen pada spektrum ke dalam bilangan bulat positif 1, 2, 3, …, |*S*|, dimana |*S*|

menyatakan kardinalitas dari spektrum *S*. Tiap kode dari fragmen dapat dianggap sebagai gen pada kromosom, sehingga kromosom adalah untaian dari kode-kode fragmen pada spektrum yang tidak berulang.

Populasi awal dibentuk dengan cara memilih kromosom (individu) secara acak sebanyak ukuran populasi yang diinginkan. Ukuran populasi berpengaruh pada daerah pencarian solusi dalam algoritma genetik serta waktu komputasi yang dibutuhkan. Semakin besar ukuran populasi, semakin luas daerah pencarian solusi sehingga semakin baik solusi yang diperoleh, tetapi semakin lama waktu komputasi yang dibutuhkan. Ukuran populasi sebaiknya disesuaikan dengan ukuran masalah yang akan diselesaikan. Hal ini karena jika ukuran masalah besar dan ukuran populasi terlalu kecil maka solusi yang diperoleh cenderung kurang baik, sebaliknya jika ukuran masalah kecil dan ukuran populasi terlalu besar, maka akan terjadi pemborosan waktu komputasi.

2.2.2 Seleksi

Setelah membentuk populasi awal, langkah selanjutnya adalah melakukan seleksi pada populasi tersebut. Proses seleksi ini dilakukan dengan tujuan mempertahankan individu dengan kualitas baik dan mengeliminasi individu dengan kualitas buruk dari generasi ke generasi berikutnya. Alat yang biasa digunakan untuk melakukan seleksi adalah fungsi

fitness. Fungsi *fitness* merupakan fungsi objektif yang digunakan untuk mengukur keoptimalan dari suatu solusi. Fungsi *fitness* yang ideal adalah yang cukup berkorelasi dengan tujuan dari algoritma genetik yang digunakan dan tidak sulit untuk dievaluasi.

Individu yang lolos proses seleksi adalah individu dengan nilai *fitness* terbaik. Hal tersebut karena diharapkan dari individu yang baik akan didapatkan keturunan yang baik bahkan lebih baik dari individu tersebut sesuai dengan prinsip evolusi biologi. Terdapat beberapa metode seleksi yang biasa digunakan, yaitu: *deterministic selection*, *roulette-wheel*, *truncation*, dan *tournament* [MA 09]. *Deterministic selection* adalah metode seleksi yang memilih sejumlah individu dengan nilai fitness terbaik dari populasi. *Roulette-wheel* adalah suatu metode seleksi yang mengaitkan kemungkinan terpilihnya suatu kromosom dengan kekuatannya (biasanya diukur berdasarkan nilai *fitness*). Pada metode ini, kemungkinan suatu kromosom terpilih sebanding dengan kekuatannya. *Truncation* adalah suatu metode seleksi dimana hanya sebagian dari kromosom terkuat yang memiliki kemungkinan untuk terpilih secara acak untuk masuk ke dalam tahap berikutnya. *Tournament* adalah metode seleksi dengan cara membuat kelompok yang terdiri dari kromosom-kromosom secara acak kemudian kromosom yang paling baik dari kelompok ini akan dipilih untuk masuk ke tahap berikutnya. Namun, kadang proses seleksi menggunakan suatu metode yang merupakan gabungan dari beberapa metode yang ada. Jenis

metode seleksi yang digunakan dalam algoritma genetik pada skripsi ini adalah suatu metode yang menggabungkan antara metode *roulette-wheel* dan *deterministic selection.*

Dalam metode seleksi tersebut, proses seleksi akan menghasilkan populasi orang tua. Kemudian dari populasi orang tua akan dipilih secara acak sepasang individu orang tua yang kemudian mengalami proses *crossover*.

2.2.3 *Crossover*

Crossover adalah suatu operasi genetik yang bertujuan menghasilkan individu baru (reproduksi). Dalam proses *crossover* akan dilakukan penyilangan (perkawinan) dua kromosom yang disebut kromosom orang tua untuk menghasilkan kromosom baru yang disebut kromosom anak. Secara umum, *crossover* dilakukan dengan menukar sebagian struktur kromosom orang tua secara sederhana. Proses tersebut dilakukan pada setiap pasang kromosom orang tua yang diperoleh dari tahap seleksi. Dari proses *crossover* ini akan didapat sejumlah kandidat solusi (populasi) untuk generasi berikutnya. Dalam algoritma genetik, banyaknya proses *crossover* yang terjadi pada tiap generasi tergantung pada probabilitias *crossover* (*c*). Pada umumnya algoritma genetik memiliki probabilitas *crossover* antara 0.6 dan 1.0 [SA 04].

Terdapat beberapa metode *crossover*, yaitu: *one point*, *two point*, *K point*, dan *cut and splice* [MA 09]. *One point* adalah suatu metode *crossover* yang memilih secara acak satu titik *crossover* pada kromosom orang tua kemudian menukar segmen kedua kromosom orang tua pada titik ini untuk menghasilkan dua kromosom anak. Metode *crossover two point* mempunyai cara kerja yang sama dengan *one point*, perbedaannya terletak pada pemilihan titik *crossover*. Dalam metode ini dipilih secara acak dua titik *crossover*. Pada *one point* dan *two point*, letak titik *crossover* yang dipilih pada kedua kromosom orang tua harus sama. *K point* adalah metode *crossover* yang merupakan generalisasi dari *one point* dan *two point.* Sedangkan *cut and splice* adalah suatu metode *crossover* dengan letak titik *crossover* pada kedua kromosom orang tua tidak harus sama. Akibatnya, dua kromosom anak yang dihasilkan bisa memiliki panjang yang berbeda.

Jenis metode *crossover* yang digunakan pada algoritma genetik yang akan dibahas dalam skripsi ini tidak seperti metode *crossover* pada umumnya. Tiap dua kromosom orang tua akan menghasilkan satu kromosom anak. Dalam pembentukan kromosom anak, terdapat struktur tertentu yang harus diperhatikan. Metode tersebut dinamakan *structured crossover*.

Dalam algoritma genetik, tahap setelah proses *crossover* adalah tahap dilakukannya proses mutasi.

2.2.4 Mutasi

Apabila dalam suatu populasi, kromosom yang satu dengan yang lainnya terlalu mirip, maka dapat menyebabkan populasi akan terjebak pada solusi optimal lokal. Oleh sebab itu, untuk menghindari kondisi tersebut, keanekaragaman genetik pada suatu populasi dari suatu generasi ke generasi berikutnya harus dipertahankan. Operator genetik yang biasa digunakan untuk melakukan hal tersebut adalah operator mutasi.

Prinsip kerja operator mutasi adalah mengubah satu atau lebih posisi gen pada suatu kromosom sehingga dihasilkan kromosom yang lebih bervariasi. Kromosom baru yang terbentuk ini diharapkan akan menghasilkan solusi yang lebih baik dari kandidat solusi sebelumnya. Seperti halnya proses *crossover,* banyaknya proses mutasi yang terjadi pada tiap generasi juga tergantung pada probabilitas mutasi (*m*). Pada umumnya algoritma genetik memiliki probabilitas mutasi yang sangat rendah.

Terdapat beberapa metode mutasi, antara lain: *swap mutation*, *flip bit*, *addition*, dan *Gaussian change* [MA 09]. Metode mutasi yang digunakan dalam algoritma genetik pada skripsi ini adalah *swap mutation*. *Swap mutation* adalah suatu metode mutasi yang menukar posisi dua buah gen yang terpilih secara acak pada suatu kromosom (individu). Seperti yang telah dijelaskan pada Subbab.2.2.1 bahwa kromosom direpresentasikan sebagai untaian dari fragmen-fragmen anggota spektrum yang telah dikodekan ke

dalam bilangan bulat positif 1, 2, 3,…, |*S*|. Dengan menganggap tiap fragmen sebagai gen, proses mutasi dilakukan dengan cara menukar posisi dua fragmen. Pada skripsi ini dua fragmen yang akan ditukar posisinya ditentukan berdasarkan aturan yang akan dijelaskan pada Bab III.

2.2.5 Pembentukan Populasi untuk Generasi Berikutnya

Seperti telah dijelaskan sebelumnya bahwa proses seleksi, *crossover*, dan mutasi dalam algoritma genetik dilakukan secara berulang pada populasi di tiap generasi. Terdapat beberapa aturan yang biasa digunakan untuk menentukan populasi yang akan digunakan untuk generasi berikutnya. Contoh aturan yang biasa digunakan adalah populasi anak baik yang dihasilkan dari proses *crossover* maupun mutasi akan menjadi generasi berikutnya menggantikan populasi orang tua. Contoh lainnya adalah aturan yang digunakan pada skripsi ini, yakni generasi berikutnya terdiri dari populasi anak yang dihasilkan dari proses *crossover* dan sebagian populasi orang tua dengan kualitas terbaik yang diukur berdasarkan nilai *fitness*nya.

2.2.6 Kriteria Berhenti (*Stopping Criteria*)

Sebelumnya telah dijelaskan bahwa secara umum proses kerja dari algoritma genetik meliputi pemilihan suatu populasi awal, mengevaluasi

populasi tersebut dengan suatu fungsi *fitness*, melakukan proses seleksi, dan selanjutnya menerapkan prinsip-prinsip evolusi dengan menggunakan operator-operator genetik seperti *crossover* dan mutasi, dan mengulangi tahapan tersebut hingga mencapai kriteria berhenti.

Terdapat lima jenis kriteria berhenti yang biasa digunakan dalam algoritma genetik, yakni: *generations*, *time limit*, *fitness limit*, *stall generations*, dan *stall time limit* [MA 09]. *Generations*, algoritma genetik berhenti ketika generasi populasi yang terbentuk mencapai batas generasi yang telah ditentukan. *Time limit*, algoritma genetik berhenti ketika waktu komputasi yang dihabiskan mencapai batas waktu yang telah ditentukan. *Fitness limit*, algoritma genetik berhenti ketika nilai *fitness* terbaik dari populasi kurang dari (untuk masalah meminimumkan) atau lebih dari (untuk masalah memaksimumkan) atau sama dengan batas nilai *fitness* yang telah ditentukan. *Stall generations*, algoritma genetik berhenti ketika tidak terjadi perubahan yang signifikan pada nilai fungsi *fitness* pada barisan populasi selama panjang selang generasi yang telah ditentukan. Misalkan panjang interval generasi yang ditentukan adalah 5 generasi. Jika selama 5 generasi tidak terjadi perubahan yang signifikan pada nilai fungsi *fitness* pada populasi di generasi tersebut, maka algoritma genetik akan berhenti bekerja. Jenis kriteria berhenti lainnya yang biasa digunakan dalam algoritma genetik adalah *stall time limit*, yakni suatu jenis kriteria berhenti dimana algoritma

genetik berhenti apabila tidak terjadi perubahan yang signifikan pada nilai fungsi *fitness* dalam selang waktu yang panjangnya telah ditentukan.

Pada skripsi akan digunakan jenis kriteria berhenti *generations.* Hal tersebut karena kriteria berhenti *generations* merupakan kriteria berhenti yang dinilai cukup baik untuk penggunaan algoritma genetik pada DNA SBH [JGKM 06].

BAB III

PENERAPAN ALGORITMA GENETIK DALAM *DNA SEQUENCING BY HYBRIDIZATION*

Pada bab ini akan dibahas mengenai penggunaan algoritma genetik dalam melakukan DNA SBH. Tujuan dari algoritma ini, dari suatu spektrum *S* akan dicari suatu barisan DNA dengan panjang tertentu sedemikian sehingga *error* positif dan *error* negatif dari pembentukan barisan DNA minimum. Dalam algoritma ini, individu atau kromosom yang terbentuk kadang berkorespondensi dengan barisan DNA yang panjangnya lebih dari panjang barisan DNA yang akan dicari (*n*). Namun demikian, *subsequence* dengan panjang tidak lebih dari *n* akan dipilih menjadi solusi yang mungkin. Jadi, masukan (*input*) dari algoritma ini adalah spektrum (*S)* dan panjang dari barisan DNA yang akan dicari (*n*). Sedangkan keluaran (*output*) adalah urutan fragmen sedemikian sehingga barisan DNA yang terbentuk memiliki *subsequence* sepanjang *n* dengan *error* positif dan *error* negatif minimum. Spektrum yang digunakan sebagai masukan merupakan spektrum ideal, yakni spektrum tanpa *error* positif dan *error* negatif. Selain itu, spektrum yang digunakan merupakan spektrum hasil hibridisasi yang menggunakan prinsip

isometric oligonucleotide library, yakni spektrum yang beranggotakan fragmen dengan panjang sama.

Representasi kromosom dan pembentukan populasi awal pada algoritma genetik akan dibahas pada Subbab 3.1. Evaluasi fungsi *fitness* dari tiap kromosom akan dibahas pada Subbab 3.2. Kemudian tahap terpenting pada algoritma yakni tahap evolusi yang melibatkan operator evolusi seleksi, *crossover*, dan mutasi masing-masing akan dibahas pada Subbab 3.3, 3.4, dan 3.5. Pembentukan populasi untuk generasi berikutnya akan dibahas pada Subbab 3.6. Subbab 3.7 akan membahas mengenai kriteria berhenti yang digunakan. Sedangkan algoritma genetik untuk DNA SBH akan diberikan pada Subbab 3.8.

3.1 REPRESENTASI KROMOSOM DAN PEMBENTUKAN POPULASI AWAL

Jenis representasi kromosom dalam algoritma genetik ini adalah *representation in multiple matters* yakni tiap individu direpresentasikan oleh beberapa simbol atau karakter yang berbeda. Langkah awalnya adalah mengkodekan fragmen-fragmen pada spektrum ke dalam bilangan bulat positif 1, 2, 3,…, $|S|$, dimana $|S|$ menyatakan kardinalitas dari spektrum S. Tiap kode dari fragmen dapat dianggap sebagai gen pada kromosom,

sehingga kromosom adalah untaian dari kode-kode fragmen pada spektrum yang tidak berulang. Tiap kromosom atau individu berkorespondensi dengan suatu barisan DNA. Pembentukan barisan DNA dari suatu kromosom memperhatikan tumpang tindih antara fragmen yang bertetangga.

Populasi awal terdiri dari *N* kromosom, dengan *N* didefinisikan sebagai setengah dari panjang barisan DNA (*n*) yang akan dicari [JGKM 06]. Setiap kromosom dibentuk dengan cara membuat permutasi dari bilangan 1, 2,…,|*S*| secara acak. Populasi awal menyatakan kumpulan solusi awal yang mungkin. Banyaknya populasi pada tiap generasi dibuat selalu sama dengan banyaknya populasi awal. Pada Contoh 3.1 diberikan contoh pembentukan kromosom dan populasi awal dari suatu masalah DNA SBH.

Tabel 3.1 Kode fragmen dalam bilangan bulat

Fragmen	Kode
ATA	1
TAA	2
AAC	3
ACG	4
CGT	5
GTA	6
TAT	7
ATT	8

Contoh 3.1

Diberikan spektrum *S* = {*ATA, TAA, AAC, ACG, CGT, GTA, TAT, ATT*} dan panjang barisan DNA yang akan dicari adalah 10. Barisan DNA asal dari spektrum *S* adalah barisan DNA *ATAACGTATT*. Pertama-tama tiap anggota

spektrum tersebut dikodekan ke dalam bilangan bulat 1, 2, 3,…, 8 seperti yang terlihat pada Tabel 3.1. Karena panjang barisan DNA yang akan dicari adalah 10 maka banyaknya kromosom dalam populasinya adalah setengah dari 10, yakni 5. Misalkan kelima kromosom yang terbentuk adalah seperti yang terlihat pada Tabel 3.2.

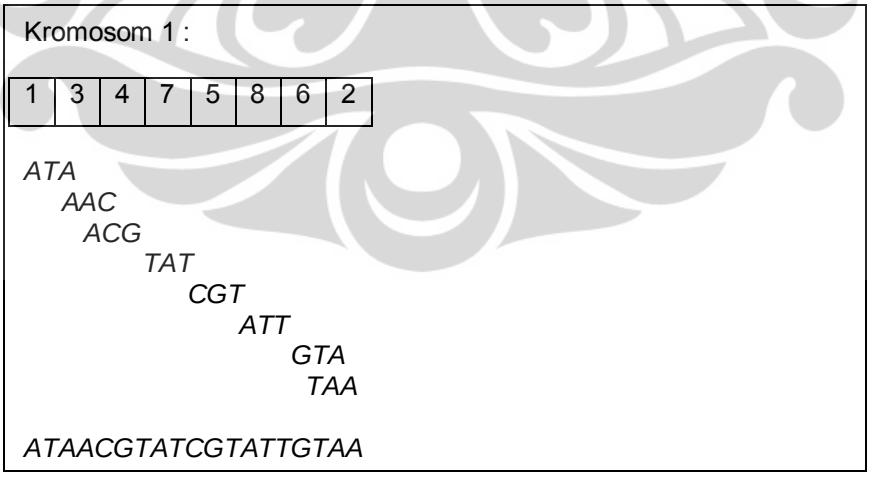
Tabel 3.2 Populasi awal yang terbentuk

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

Setiap kromosom pada populasi berkorespondensi dengan suatu barisan DNA. Barisan DNA yang bersesuaian dengan suatu kromosom dapat diperoleh dengan mendekodekan gen 1, 2, …, $|S|$ menjadi fragmen dan merangkai fragmen sesuai urutan gen pada kromosom dengan memperhatikan nukleotida-nukleotida yang tumpang tindih pada fragmen-fragmen yang bertetangga atau berdekatan.

Proses pembentukan barisan DNA yang berkorespondensi dengan kromosom 1 dapat dilihat pada Gambar 3.1. Pada kromosom 1 yang memiliki urutan fragmen 1-3-4-7-5-8-6-2, fragmen-fragmen tersebut kemudian disusun dengan memperhatikan tumpang tindih antara fragmen yang bertetangga. Pada kromosom 1, fragmen 1 dan 3 bertetangga. Karakter terakhir fragmen 1 sama dengan karakter pertama fragmen 3, sehingga awal karakter pada

fragmen 3 diletakkan pada karakter terakhir fragmen 1. Selanjutnya, fragmen 3 bertetangga dengan fragmen 4. Dua karakter pertama fragmen 4 sama dengan 2 karakter terakhir pada fragmen 3, sehingga dua karakter pertama fragmen 4 diletakkan pada dua karakter terakhir fragmen 3. Fragmen 4 bertetangga dengan fragmen 7. Karakter terakhir fragmen 4 tidak sama dengan karakter pertama fragmen 7. Selain itu, dua karakter terakhir fragmen 4 juga tidak sama dengan dua karakter pertama pada fragmen 7, sehingga fragmen 7 diletakkan setelah karakter terakhir fragmen 4. Kemudian hal yang sama dilakukan pada pasangan fragmen 7 dan fragmen 5, fragmen 5 dan fragmen 8, fragmen 8 dan fragmen 6, serta fragmen 6 dan fragmen 2. Penyusunan fragmen seperti itu menghasilkan barisan DNA *ATAACGTATCGTATTGTAA* . Dengan cara yang sama, barisan DNA yang bersesuaian dengan kromosom 2, 3, 4, dan 5 dapat dilihat pada Tabel 3.3.



Gambar 3.1 Barisan DNA yang berkorespondensi dengan kromosom 1

Tabel 3.3 Barisan DNA yang berkorespondensi dengan masing-masing kromosom

Kromosom	Representasi Kromosom	Barisan DNA
Kromosom 1	1-3-4-7-5-8-6-2	*ATAACGTATCGTATTGTAA*
Kromosom 2	3-7-8-6-5-1-2-4	*AACTATTGTACGTATAACG*
Kromosom 3	3-8-6-1-2-4-5-7	*AACATTGTATAACGTAT*
Kromosom 4	2-1-4-5-3-7-8-6	*TAATACGTAACTATTGTA*
Kromosom 5	8-4-5-6-7-1-2-3	*ATTACGTATAAC*

3.2 EVALUASI FUNGSI *FITNESS* KROMOSOM

Fungsi *fitness* digunakan untuk menentukan seberapa baik individu yang direpresentasikan oleh suatu kromosom. Pada Subbab 2.1 telah dijelaskan bahwa tujuan dari DNA SBH adalah mencari suatu barisan DNA dengan panjang tertentu sedemikian sehingga *error* positif dan *error* negatif dari pembentukan barisan DNA minimum. Berdasarkan definisi dari *error* positif dan *error* negatif, dapat dikatakan bahwa semakin banyak fragmen di dalam spektrum yang terpakai untuk membentuk barisan DNA maka semakin minimum *error* dari barisan yang terbentuk. Oleh sebab itu, dalam algoritma genetik ini, *fitness* dari suatu kromosom didefinisikan sebagai banyaknya fragmen yang terpakai untuk membentuk *best subsequence*, yakni *subsequence* terpanjang yang panjangnya tidak lebih dari panjang barisan DNA yang akan dicari (*n*) dan mencakup fragmen terbanyak. *Subsequence* dalam hal ini didefinisikan sebagai berikut :

Misalkan o_i menyatakan fragmen pada urutan ke-i pada kromosom. Jika kromosom memiliki urutan fragmen sebagai berikut $o_1 o_2 o_3 ... o_{|S|}$, maka *subsequence* yang terbentuk merupakan *sequence* yang dibentuk dengan urutan fragmen $o_{k_1} o_{k_2} o_{k_3} ... o_{k_t}$ dengan $k_1 < k_2 < k_3 < ... < k_t$ $(1 \leq k_j \leq |S|, k_j \in Z^+)$. Dalam urutan fragmen pada kromosom, fragmen o_{k_j} harus merupakan ***predecessor*** dari fragmen $o_{k_{j+1}}$ yakni fragmen o_{k_j} berada pada urutan tepat sebelum fragmen $o_{k_{j+1}}$. Dengan kata lain, fragmen $o_{k_{j+1}}$ adalah ***successor*** dari fragmen o_{k_j}, yakni fragmen $o_{k_{j+1}}$ berada pada urutan tepat setelah fragmen o_{k_j}. Sebagai contoh, diberikan Contoh 3.2.

Contoh 3.2

Dari contoh 3.1 dikatakan bahwa akan dicari barisan DNA dengan panjang 10. Perhatikan urutan fragmen pada kromosom 1 pada Gambar 3.1. *Subsequence* dengan panjang tidak lebih dari 10 dan fragmen yang membentuk *subsequence* tersebut dapat dilihat pada Tabel 3.4.

Berdasarkan definisi dari *best subsequence*, maka dari Tabel 3.4 terlihat bahwa *best subsequence* dari kromosom 1 adalah *AACGTATCGT* dengan banyak fragmen yang membentuk *subsequence* tersebut adalah 4. Jadi, *fitness* dari kromosom 1 bernilai 4. Dengan melakukan hal yang sama pada kromosom 2, 3, 4, dan 5, maka didapat Tabel 3.5.

Tabel 3.4 *Subsequence* pada kromosom 1 dengan panjang tidak lebih dari 10

Urutan Fragmen	*Subsequence*	Panjang *Subsequence*	Banyak Fragmen
1-3-4-7	*ATAACGTAT*	9	4
3-4-7-5	*AACGTATCGT*	10	4
4-7-5	*ACGTATCGT*	9	3
7-5-8	*TATCGTATT*	9	3
5-8-6	*CGTATTGTA*	9	3
8-6-2	*ATTGTAA*	7	3

Tabel 3.5 Nilai *fitness* dari masing-masing kromosom

Kromosom	Nilai *Fitness* (F)
1	4
2	4
3	5
4	4
5	7
Jumlah	24

3.3 SELEKSI

Seleksi adalah suatu proses untuk memilih individu-individu terbaik pada populasi yang akan digunakan pada proses selanjutnya, yakni proses *crossover*. Pada Subbab 2.2.2 telah dijelaskan bahwa metode seleksi yang digunakan dalam algoritma genetik ini menggabungkan antara prinsip metode seleksi *roulette wheel* dan *deterministic*. *Deterministic selection* adalah metode seleksi yang memilih sejumlah individu dengan nilai fitness terbaik dari populasi. *Roulette-wheel* adalah suatu metode seleksi yang mengaitkan kemungkinan terpilihnya suatu kromosom dengan kekuatannya (biasanya diukur berdasarkan nilai *fitness*).

Dalam metode *roulette-wheel*, proses seleksi individu diibaratkan seperti dalam permainan judi *roulette-wheel*. Dalam permainan *roulette-wheel,* pemain akan memutar roda yang telah terpartisi menjadi beberapa bagian dengan tiap bagian bersesuaian dengan suatu hadiah. Pada penyangga roda terdapat sebuah jarum penunjuk yang akan menentukan hadiah apa yang akan didapat oleh pemain. Hadiah yang didapat oleh pemain adalah hadiah yang ditunjuk oleh jarum penunjuk ketika roda berhenti berputar. Dalam sekali putaran roda, pemain hanya mempunyai kemungkinan mendapat satu hadiah. Kaitannya dengan metode seleksi yang dibahas ini adalah suatu kromosom diibaratkan sebagai hadiah. Nilai harapan tiap kromosom dihitung dengan suatu cara sedemikian sehingga menyerupai kondisi seperti pada roda *roulette-wheel*. Kemudian proses memutar roda dinyatakan dengan menentukan suatu bilangan acak (*random)* pada interval (0,1]. Namun tidak seperti pada permainan *roulette-wheel* dimana dalam sekali putaran pemain hanya mendapat satu hadiah, dalam metode seleksi ini, dalam sekali proses seleksi didapatkan lebih dari satu kromosom, yakni sebanyak ukuran populasi. Dalam metode seleksi ini, suatu kromosom kadang terpilih sebanyak lebih dari sekali. Kromosom-kromosom yang terpilih tersebut akan membentuk populasi orang tua. Selanjutnya, kromosom-kromosom yang akan digunakan pada proses *crossover* akan diambil dari populasi orang tua secara acak.

Berdasarkan prinsip proses seleksi yang telah dijelaskan, maka tahapan proses seleksi adalah sebagai berikut :

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

Tahap 2: Hitung nilai rata-rata *fitness* dari semua kromosom. Jika F_i menyatakan nilai *fitness* dari kromosom ke-i , $i = 1, 2, ..., N$ (N menyatakan ukuran populasi), maka nilai rata-rata *fitness*nya adalah

$$\overline{F} = \frac{\sum_{i=1}^{N} F_i}{N}$$

Tahap 3: Pilih secara acak bilangan r pada interval (0,1].

Tahap 4: Hitung nilai harapan untuk tiap kromosom, yang dinyatakan dengan array fp , dimana

$$fp[1] = F_1 / \overline{F} \qquad fp[i] = F_i / \overline{F} + fp[i-1] \quad i = 2,3....., N$$

Tahap 5: Pemilihan kromosom dengan algoritma sebagai berikut.

```
begin
  i = 1
  count = 1
  while count ≤ N do
      If r < fp[i] do
            select = 1 (kromosom ke-i dipilih)
            r = r + 1
            count = count + 1
      else
            select = 0 (kromosom ke-i tidak dipilih)
            i = i + 1
      end if;
  end while;
end;
```

Contoh proses seleksi diberikan pada Contoh 3.3.

Tabel 3.6 Nilai harapan($fp[i]$) dari masing-masing kromosom

i	F_i	$fp[i]$
1	4	5/6
2	4	10/6
3	5	65/24
4	4	85/24
5	7	5

Contoh 3.3

Dengan menggunakan Tabel 3.5 pada Contoh 3.2 akan dihitung nilai $fp[i]$ dari masing-masing kromosom ke-i . Pertama-tama akan dihitung nilai rata-rata *fitness* ($\overline{F}$). Misalkan $\sum F$ menyatakan jumlah nilai *fitness* semua kromosom dan N menyatakan banyaknya kromosom maka $\overline{F} = \frac{\sum F}{N} = \frac{24}{5}$.

Selanjutnya akan dihitung nilai $fp[i]$ dari masing-masing kromosom ke-*i*.

- Untuk kromosom 1 ($i = 1$)

$$fp[1] = \frac{F_i}{\overline{F}} = \frac{4}{24/5} = \frac{5}{6}$$

- Untuk kromosom 2 ($i = 2$)

$$fp[2] = \frac{F_2}{\overline{F}} + fp[1] = \frac{4}{24/5} + \frac{5}{6} = \frac{5}{6} + \frac{5}{6} = \frac{10}{6}$$

Dengan melakukan hal yang sama pada kromosom 3, 4, dan 5 maka didapat Tabel 3.6. Selanjutnya akan dilakukan proses pemilihan kromosom. Misalkan bilangan *r* yang terpilih adalah *r* = 0.8.

- *i* = 1

  count = 1

  Karena $fp[1] = \frac{5}{6} > 0.8$ maka kromosom 1 dipilih.

  Kemudian hitung *r* = 0.8 + 1 = 1.8.

  count = 1 + 1 = 2

  Karena $fp[1] = \frac{5}{6} < 1.8$ maka hitung *i* = 1 + 1 = 2

- *i* = 2

  count = 2

  Karena $fp[2] = \frac{10}{6} < 1.8$ maka hitung *i* = 2 + 1 = 3

- *i* = 3

  count = 2

  Karena $fp[3] = \frac{65}{24} > 1.8$ maka kromosom 3 dipilih

  Kemudian lakukan proses tersebut secara berulang. Jika *count* > 5 maka proses pemilihan kromosom berhenti. Proses pemilihan kromosom untuk membentuk populasi orang tua terangkum pada Tabel 3.7. Berdasarkan Tabel 3.7, berarti kromosom yang terpilih menjadi anggota dari populasi orang tua adalah seperti yang tertulis pada Tabel 3.8. Selanjutnya, dari

populasi orang tua akan dipilih sepasang kromosom secara acak untuk digunakan dalam proses *crossover*.

Tabel 3.7 Proses pemilihan kromosom

i	*count*	*fp[i]*	*r*	*r<fp[i]*	*Select*	*Increment*
1	1	5/6	0.8	*true*	*yes*	*r, count*
1	2	5/6	1.8	*false*	*no*	*i*
2	2	10/6	1.8	*false*	*no*	*i*
3	2	65/24	1.8	*true*	*yes*	*r, count*
3	3	65/24	2.8	*false*	*no*	*i*
4	3	85/24	2.8	*true*	*yes*	*r, count*
4	4	85/24	3.8	*false*	*no*	*i*
5	4	5	3.8	*true*	*yes*	*r, count*
5	5	5	4.8	*true*	*yes*	*r, count*
5	6	5	5.25	***Proses berhenti***		

Tabel 3.8 Populasi orang tua

Orang tua	Kromosom	Representasi Kromosom
Orang tua 1	kromosom 1	1-3-4-7-5-8-6-2
Orang tua 2	kromosom 3	3-8-6-1-2-4-5-7
Orang tua 3	kromosom 4	2-1-4-5-3-7-8-6
Orang tua 4	kromosom 5	8-4-5-6-7-1-2-3
Orang tua 5	kromosom 5	8-4-5-6-7-1-2-3

3.4 *CROSSOVER*

Crossover merupakan suatu proses persilangan sepasang kromosom orang tua untuk menghasilkan kromosom anak yang akan menjadi individu pada populasi di generasi berikutnya. Kromosom anak yang dihasilkan dari proses *crossover* diharapkan mewarisi sifat-sifat unggul yang dimiliki oleh

kromosom orang tua. Sebelumnya telah dijelaskan bahwa semakin banyak fragmen anggota spektrum yang terpakai untuk membentuk barisan maka semakin minimum *error* barisan yang terbentuk. Dalam pembentukan barisan DNA, semakin banyak tumpang tindih yang terjadi antara fragmen maka semakin banyak fragmen anggota spektrum yang terpakai untuk membentuk barisan DNA tersebut. Oleh sebab itu, kondisi tumpang tindih atau *overlap* menjadi sangat penting dalam menentukan solusi.

Berdasarkan penjelasan pada Subbab 2.2.3, metode *crossover* yang digunakan adalah *structured crossover* yakni dalam pembentukan kromosom anak terdapat struktur tertentu yang harus diperhatikan. Struktur yang diperhatikan adalah tumpang tindih antara fragmen-fragmen yang bertetangga dan susunan fragmen pada kromosom orang tua. Sebelum menjelaskan tentang proses *crossover*, terlebih dahulu akan dijelaskan mengenai ***overlap degree*** (*od*), ***total overlap degree*** (*tod*), dan suatu fungsi yang disebut dengan fungsi ***fits***. Yang dimaksud *overlap degree* (*od*) adalah banyaknya nukleotida yang saling tumpang tindih antara dua fragmen yang bertetangga. Nilai *od* antara dua fragmen yang bertetangga dinotasikan dengan $od(o_i, o_j)$ dengan o_i adalah *predecessor* dari o_j. Sedangkan yang dimaksud dengan *total overlap degree* (*tod*) adalah jumlah *overlap degree* dari setiap pasang fragmen yang saling bertetangga. Nilai *tod* dari beberapa fragmen yang bertetangga dinotasikan dengan

$tod\ (o_1, o_2, ..., o_k) = \sum_{i=1}^{k-1} od\ (o_i, o_{i+1})$ dengan o_i merupakan *predecessor* dari o_{i+1}.

Untuk memahami istilah *overlap degree* (*od*) dan *total overlap degree* (*tod*) diberikan Contoh 3.4.

Contoh 3.4

Diberikan tiga fragmen dengan urutan seperti pada Gambar 3.2.

ATA	*od*(*ATA, AAC*) = 1
AAC	
ACG	*od*(*AAC, ACG*) = 2
ATAACG	*tod*(*ATA,AAC,ACG*) = 3

Gambar 3.2 Contoh *overlap degree* dan *total overlap degree*

Pada susunan fragmen pada Gambar 3.2 terlihat bahwa karakter terakhir fragmen *ATA* sama dengan karakter pertama fragmen *AAC*. Hal ini berarti *od*(*ATA, AAC*) = 1. Selain itu pada pasangan fragmen *AAC* dan *ACG* terlihat bahwa dua karakter pertama fragmen *ACG* sama dengan dua karakter terakhir fragmen *AAC*. Hal ini berarti didapat *od*(*AAC, ACG*) = 2. Jadi, *tod*(*ATA,AAC,ACG*) = 3.

Selanjutnya akan dibahas mengenai fungsi *fits*. Fungsi *fits* digunakan saat proses *crossover*. Misalkan S menyatakan spektrum dan S^* ($S^* \subset S$) menyatakan kumpulan fragmen pada *block* kromosom anak. Yang dimaksud *block* kromosom anak adalah kromosom anak sementara atau kromosom

anak dengan gen yang belum lengkap. Kemudian misalkan $p^* \in S$ dan $p^* \notin S^*$. Jika *sequence** adalah barisan DNA yang dibentuk oleh $S^* \cup \{p^*\}$, dan $\left|S^* \cup \{p^*\}\right|$ menyatakan kardinalitas dari $S^* \cup \{p^*\}$, dan l(*sequence**) menyatakan panjang *sequence**, maka fungsi *fits* didefinisikan sebagai berikut :

$$fits(sequence^*) = \frac{\left|S^* \cup \{p^*\}\right|}{l(sequence^*)}$$

Misalkan diberikan dua barisan DNA yang masing-masing barisan dibentuk oleh himpunan fragmen yang sama tetapi dengan urutan yang berbeda. Barisan DNA yang berukuran lebih pendek menandakan urutan fragmen pada barisan DNA tersebut memberikan nilai *tod* yang lebih besar. Berdasarkan definisi fungsi *fits* yang telah dijelaskan sebelumnya terlihat bahwa semakin pendek barisan DNA yang terbentuk maka semakin besar nilai *fits.*

Kromosom anak hasil proses *crossover* diharapkan memiliki nilai *total overlap degree* yang besar dengan tujuan kromosom anak semakin mendekati solusi yang diinginkan. Dalam menghasilkan kromosom anak, diasumsikan fragmen yang pertama kali terdapat pada *block* kromosom anak sudah diketahui. Berdasarkan penjelasan sebelumnya, maka dalam menentukan urutan fragmen berikutnya, fragmen yang memberikan nilai *fits*

terbesar akan terlebih dahulu disusun pada *block* kromosom anak. Dalam proses pembentukan kromosom anak, selain nilai *fits*, hal lain yang perlu diperhatikan yakni *predecessor* dan *successor* tiap fragmen pada kromosom orang tua. Pada proses *crossover* ini, sepasang kromosom orang tua hanya akan menghasilkan satu kromosom anak.

Misalkan *c* menyatakan probabilitas *crossover* dan *N* menyatakan ukuran populasi, maka banyaknya proses *crossover y*ang terjadi pada setiap generasi dalam algoritma genetik ini adalah *c* x *N* kali. Hal ini karena pada tiap generasi terdapat *N* individu yang berkemungkinan melakukan proses *crossover.*

Adapun tahapan proses *crossover* adalah sebagai berikut:

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

Tahap 2: Pilih secara acak fragmen pertama yang terdapat pada keturunan (anak), sebut sebagai o_i. Misalkan o_f (*first*) dan o_l (*last*) secara berturut-turut menyatakan fragmen pada urutan pertama dan terakhir pada suatu kromosom anak. Berarti pada tahap awal $o_f = o_i = o_l$.

Tahap 3: Identifikasi letak fragmen o_i pada kedua kromosom orang tua. Perhatikan *predecessor* dan *successor* dari o_i pada masing-masing kromosom orang tua.

Tahap 4: Kemudian hitung *fits* antara o_i dengan masing-masing *predecessor* atau *successor* yang ditemukan. Pilih fragmen yang memberikan nilai *fits* terbesar. Jika terdapat lebih dari satu fragmen maka pilih salah satu fragmen secara acak.

Tahap 5: Tentukan o_f dan o_l pada *block* kromosom anak.

Tahap 6: Identifikasi o_f dan o_l pada masing-masing kromosom orang tua. Perhatikan *predecessor* dari o_f dan *successor* dari o_l.

Tahap 7: Kemudian hitung *fits* antara barisan nukleotida pada *block* kromosom anak dengan masing-masing *predecessor* atau *successor* yang ditemukan. Pilih fragmen yang memberikan nilai *fits* terbesar. Jika terdapat lebih dari satu fragmen maka pilih salah satu secara acak. Jika semua *predecessor* dan *successor* sudah terdapat dalam kromosom anak, maka pilih fragmen yang belum terdapat pada kromosom anak dan menghasilkan nilai *od* terbesar terhadap o_f atau o_l .

Ulangi tahap 5 sampai dengan tahap 7 hingga semua fragmen terdapat pada kromosom anak.

Contoh proses *crossover* diberikan pada Contoh 3.5.

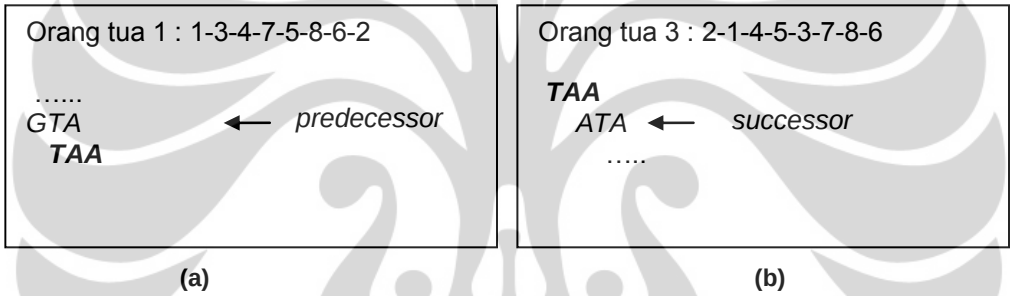
Contoh 3.5

Dengan menggunakan populasi orang tua pada Tabel 3.8, berikut akan dibahas mengenai proses *crossover* yang terjadi. Misalkan probabilitas

crossover bernilai 0.6. Karena diketahui bahwa banyaknya populasi adalah 5, maka pada tiap generasi, proses *crossover* akan terjadi sebanyak 0.6 x 5 = 3 kali.

Tahap 1: Misalkan dua kromosom yang terpilih adalah kromosom orang tua 1 dan orang tua 3 pada populasi orang tua pada Tabel 3.6.

Tahap 2: Misalkan fragmen yang pertama kali terdapat pada kromosom anak 1 adalah fragmen 2 yakni fragmen *TAA*.



Gambar 3.3 *Predecessor* dan *successor* o_i pada orang tua 1 (a) dan orang tua 3 (b)

Tahap 3: Identifikasi letak fragmen 2 pada kedua kromosom orang tua dan identifikasi *predecessor* dan *successor* dari fragmen *TAA* pada masing-masing kromosom orang tua. Pada kromosom orang tua 1 didapat *predecessor* dari fragmen 2 adalah fragmen 6 (*GTA*) yang akan membentuk barisan DNA *GTAA,* sedangkan *successor* tidak ada. Pada kromosom orang tua 3 didapat *successor* dari fragmen 2 adalah fragmen 1 (*ATA*) yang akan membentuk barisan DNA *TAATA*, sedangkan *predecessor* tidak ada.

Tahap 4: Selanjutnya akan dihitung nilai *fits*.

$$fits\ (GTAA\ ) = \frac{|\{GTA\ ,TAA\ \}|}{I(GTAA\ )} = \frac{2}{4} = \frac{1}{2}$$

$$fits\ (TAATA\ ) = \frac{|\{TAA\ ,ATA\ \}|}{I(TAATA\ )} = \frac{2}{5}$$

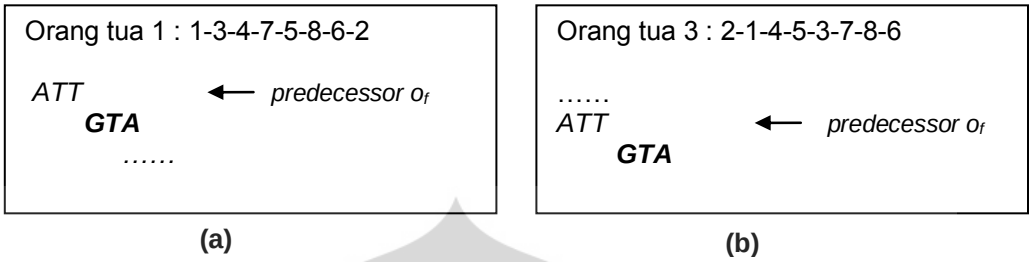
Dari hasil perhitungan di atas berarti fragmen yang dipilih adalah fragmen *GTA* dengan urutan fragmen pada *block* kromosom 1 anak menjadi seperti pada Gambar 3.4.

Block kromosom anak 1	
GTA	o_f
TAA	o_l

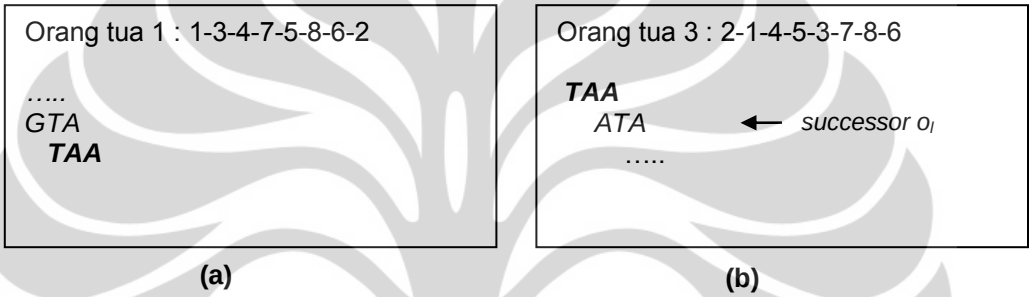
Gambar 3.4 Urutan fragmen pada *block* kromosom anak 1 setelah tahap 4

Tahap 5: Pada Gambar 3.4 terlihat bahwa o_f dan o_l pada *block* kromosom anak 1 berturut-turut adalah fragmen *GTA* dan *TAA*.

Tahap 6: Selanjutnya mengidentifikasi letak o_f dan o_l serta *predecessor* dari o_f dan *successor* dari o_l pada masing-masing kromosom orang tua seperti yang terlihat pada Gambar 3.5 dan Gambar 3.6. Pada orang tua 1 dan pada orang tua 3, *predecessor* dari fragmen 6 adalah fragmen 8 (*ATT*) yang akan membentuk barisan *ATTGTAA.*



Gambar 3.5 *Predecessor* dari fragmen 6 pada orang tua 1 (a) dan orang tua 3 (b)



Gambar 3.6 *Successor* dari fragmen 2 pada orang tua 1 (a) dan orang tua 3 (b)

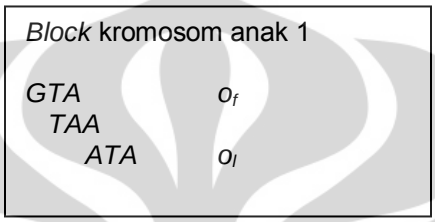
Pada Gambar 3.6 terlihat bahwa pada orang tua 1 tidak terdapat *successor* dari fragmen 2, sedangkan pada orang tua 3, *successor* dari fragmen 2 adalah fragmen 1 yang akan membentuk barisan *GTAATA.*

Tahap 7: Kemudian akan dihitung nilai *fits*.

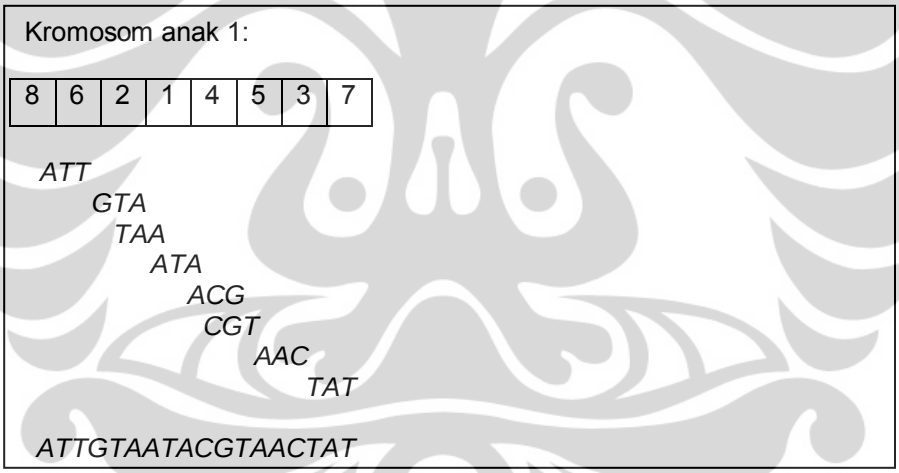
$$fits\ (ATTGTAA\quad) = \frac{\left|\{ATT\ ,GTA\ ,TAA\ \}\right|}{l(ATTGTAA\quad)} = \frac{3}{7}$$

$$fits\ (GTAATA\quad) = \frac{\left|\{GTA\ ,TAA\ ,ATA\ \}\right|}{l(GTAATA\quad)} = \frac{3}{6}$$

Dari hasil perhitungan yang diperoleh, berarti fragmen yang dipilih adalah fragmen *ATA*. Sehingga urutan fragmen pada *block* kromosom anak 1 menjadi seperti pada Gambar 3.7.



Gambar 3.7 Urutan fragmen pada *block* kromosom anak 1 setelah tahap 7



Gambar 3.8 Barisan DNA yang berkorespondensi dengan urutan fragmen pada kromosom anak 1

Dengan melakukan tahap 5 sampai tahap 7 secara berulang sampai semua fragmen terdapat dalam kromosom anak 1, maka didapat kromosom anak 1 dengan urutan fragmen seperti pada Gambar 3.8.

Proses *crossover* tersebut dilakukan sebanyak 3 kali. Misalkan populasi setelah terjadi proses *crossover* adalah seperti yang terlihat pada

Tabel 3.9. Tahapan setelah proses *crossover* yakni dilakukannya proses mutasi.

Tabel 3.9 Populasi setelah terjadi proses *crossover*

Kromosom	Representasi Kromosom
Orang tua 1	1-3-4-7-5-8-6-2
Orang tua 2	3-8-6-1-2-4-5-7
Orang tua 3	2-1-4-5-3-7-8-6
Orang tua 4	8-4-5-6-7-1-2-3
Orang tua 5	8-4-5-6-7-1-2-3
Anak 1	8-6-2-1-4-5-3-7
Anak 2	6-2-1-4-5-3-7-8
Anak 3	6-1-3-4-5-7-8-2

3.5 MUTASI

Mutasi adalah suatu proses yang dilakukan untuk mempertahankan keanekaragaman genetik populasi. Hal tersebut dilakukan untuk mencegah populasi terjebak dalam solusi optimal lokal. Dalam algoritma genetik ini, proses mutasi dapat terjadi baik pada populasi orang tua maupun populasi anak yang dihasilkan dari proses *crossover*. Metode mutasi yang digunakan adalah *swap mutation*. Seperti dalam proses *crossover*, tumpang tindih antara fragmen yang bertetangga menjadi hal yang harus diperhatikan. Prinsip dari proses mutasi ini adalah kondisi tumpang tindih pada kromosom setelah mengalami mutasi diharapkan lebih banyak dibandingkan sebelum mengalami mutasi. Proses mutasi dilakukan dengan cara menukar posisi

atau letak suatu fragmen pada kromosom sehingga urutan fragmen yang baru mempunyai kondisi tumpang tindih yang lebih banyak.

Misalkan *m* menyatakan probabilitas terjadinya mutasi, *N* menyatakan ukuran populasi, dan |*S*| menyatakan kardinalitas dari spektrum, maka banyaknya proses mutasi yang terjadi pada tiap generasi adalah *m* x *N* x |*S*|. Hal ini karena pada tiap generasi terdapat *N* kromosom dan pada tiap individu terdapat |*S*| fragmen yang berkemungkinan akan mengalami perubahan letak pada kromosom. Adapun tahapan mutasinya adalah sebagai berikut :

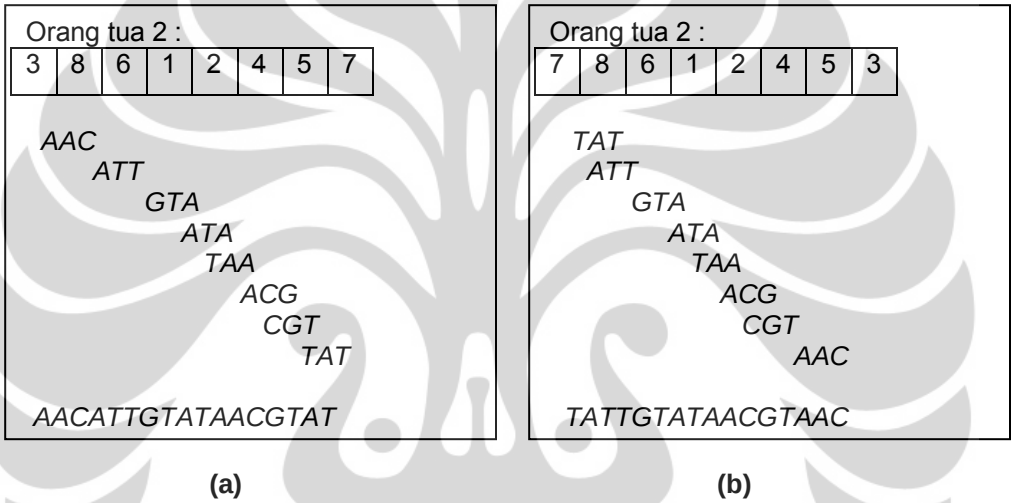
Tahap 1: Pilih satu kromosom dari populasi orang tua atau populasi anak secara acak.

Tahap 2: Hitung nilai *tod*(pred o_i, o_i, suc o_i) untuk tiap fragmen o_i. Pilih fragmen dengan nilai *tod*(pred o_i, o_i, suc o_i) terkecil. Jika terdapat lebih dari satu fragmen dengan nilai *tod*(pred o_i, o_i, suc o_i) terkecil, maka dari fragmen-fragmen tersebut pilih fragmen yang pertama kali ditemukan.

Tahap 3: Pada fragmen yang didapat pada tahap 2, misalkan fragmen tersebut dinyatakan dengan o_m, tukar posisi o_m dengan *predecessor* atau *successor*nya yang memberikan nilai *od* terkecil. Jika ternyata *od* antara o_m dengan *predecessor* dan *successor* bernilai sama maka pilih secara acak antara *predecessor* atau *successor* yang akan bertukar posisi dengan

o_m. Jika fragmen yang terpilih merupakan fragmen dengan urutan pertama (terakhir) pada kromosom, maka pindahkan posisi fragmen yang terpilih tersebut ke posisi urutan terakhir (pertama).

Contoh proses mutasi diberikan pada Contoh 3.6.



Gambar 3.9 Urutan fragmen pada kromosom orang tua 2 sebelum (a) dan sesudah (b) mengalami mutasi

Contoh 3.6

Dengan menggunakan kromosom-kromosom pada Tabel 3.9, misalkan probabilitas mutasinya adalah 0.02. Berarti proses mutasi akan terjadi sebanyak 0.02 x 5 x 8 = 0.8 kali atau dengan melakukan pembulatan menjadi sebanyak 1 kali. Misalkan kromosom yang terpilih untuk mengalami mutasi adalah kromosom orang tua 2. Urutan fragmen pada kromosom orang tua 2 sebelum mengalami mutasi dapat dilihat pada Gambar 3.9 (a). Langkah

selanjutnya adalah menghitung nilai *tod*(pred o_i, o_i, suc o_i) masing-masing fragmen. Nilai *tod*(pred o_i, o_i, suc o_i) masing-masing fragmen dapat dilihat pada Tabel 3.10.

Tabel 3.10 Nilai *tod*(pred o_i, o_i, suc o_i) untuk masing-masing fragmen o_i

Fragmen (o_i)	*Predecessor* (pred o_i)	*Successor* (suc o_i)	*od*(pred o_i, o_i)	*od*(o_i, suc o_i,)	*tod*(pred o_i,o_i,suc o_i)
AAC	-	*ATT*	-	0	0
ATT	*AAC*	*GTA*	0	0	0
GTA	*ATT*	*ATA*	0	1	1
ATA	*GTA*	*TAA*	1	2	3
TAA	*ATA*	*ACG*	2	1	3
ACG	*TAA*	*CGT*	1	2	3
CGT	*ACG*	*TAT*	2	1	3
TAT	*CGT*	-	1	-	1

Pada Tabel 3.10 terlihat bahwa nilai *tod*(pred o_i, o_i, suc o_i) terkecil adalah 0. Fragmen dengan *tod*(pred o_i, o_i, suc o_i) = 0 adalah fragmen *AAC* dan *ATT*. Karena fragmen *AAC* merupakan fragmen yang pertama kali ditemukan, maka fragmen yang dipilih adalah fragmen *AAC*. Karena fragmen *AAC* merupakan fragmen pada urutan pertama pada orang tua 2, maka langkah selanjutnya adalah menukar posisi atau urutan fragmen *AAC* dengan *TAT* yang merupakan fragmen pada urutan terakhir. Sehingga, urutan fragmen pada kromosom orang tua 2 menjadi seperti Gambar 3.9 (b). Populasi setelah proses mutasi adalah seperti yang terlihat pada Tabel 3.11.

Tabel 3.11 Populasi setelah terjadi proses mutasi

Kromosom	Representasi Kromosom
Orang tua 1	1-3-4-7-5-8-6-2
Orang tua 2	7-8-6-1-2-4-5-3
Orang tua 3	2-1-4-5-3-7-8-6
Orang tua 4	8-4-5-6-7-1-2-3
Orang tua 5	8-4-5-6-7-1-2-3
Anak 1	8-6-2-1-4-5-3-7
Anak 2	6-2-1-4-5-3-7-8
Anak 3	6-1-3-4-5-7-8-2

3.6 PEMBENTUKAN POPULASI UNTUK GENERASI BERIKUTNYA

Pada algoritma genetik ini, semua kromosom anak yang dihasilkan dari proses *crossover* baik yang belum maupun yang telah mengalami mutasi akan digunakan sebagai bagian dari populasi untuk generasi berikutnya. Selain itu, beberapa kromosom orang tua baik yang belum maupun yang telah mengalami mutasi akan menjadi bagian populasi untuk generasi berikutnya. Kromosom orang tua yang terpilih adalah kromosom dengan kualitas terbaik yang diukur berdasarkan nilai *fitness*-nya. Langkah awal dari pemilihan individu untuk populasi generasi berikutnya adalah menghitung nilai *fitness* semua kromosom anak dan orang tua. Nilai *fitness* kromosom orang tua dihitung kembali karena alasan kemungkinan telah terjadi mutasi pada kromosom orang tua. Sebagai contoh, diberikan Contoh 3.7.

Tabel 3.12 Nilai *fitness* dari setiap kromosom pada akhir generasi 1

Kromosom	Nilai *Fitness* (F)
Orang tua 1	4
Orang tua 2	5
Orang tua 3	4
Orang tua 4	7
Orang tua 5	7
Anak 1	5
Anak 2	5
Anak 3	6

Contoh 3.7

Dengan menggunakan data pada Tabel 3.11, terlihat bahwa urutan fragmen pada kromosom orang tua 1, 3, 4, dan 5 masih sama seperti urutan fragmen ketika pembentukan kromosom (tidak mengalami mutasi). Sehingga, tidak diperlukan untuk menghitung ulang nilai *fitness* dari kromosom orang tua 1, 3, 4, dan 5. Karena kromosom orang tua 2 telah mengalami mutasi maka nilai *fitness* dari kromosom orang tua 2 harus dihitung kembali. Dengan menggunakan cara seperti yang telah dijelaskan pada Subbab 3.2 maka didapat nilai *fitness* dari kromosom anak 1, anak 2, anak 3, dan orang tua 2 secara berturut-turut adalah 5, 5, 6, dan 5.

Tabel 3.13 Populasi untuk generasi berikutnya

Kromosom	Representasi Kromosom
Orang tua 4	8-4-5-6-7-1-2-3
Orang tua 5	8-4-5-6-7-1-2-3
Anak 1	3-7-8-6-5-1-2-4
Anak 2	6-2-1-4-5-3-7-8
Anak 3	6-1-3-4-5-7-8-2

Berdasarkan Tabel 3.12, maka individu yang akan digunakan untuk membentuk populasi generasi berikutnya adalah kromosom anak 1, anak 2, anak 3, orang tua 4, dan orang tua 5 seperti yang terlihat pada Tabel 3.13.

3.7 KRITERIA BERHENTI (*STOPPING CRITERIA*)

Proses-proses pada algoritma genetik akan terus berulang sampai mencapai suatu kriteria berhenti tertentu. Jenis kriteria berhenti pada algoritma genetik ini adalah *generations.* Jadi, algoritma genetik akan berhenti setelah mencapai batas generasi yang telah ditentukan. Misalkan batas generasi ditentukan sampai generasi ke-50. Hal ini berarti setelah mencapai generasi ke-50 maka algoritma genetik berhenti bekerja.

3.8 ALGORITMA GENETIK UNTUK DNA *SEQUENCING BY HYBRIDIZATION*

Berdasarkan Subbab 3.1 sampai dengan Subbab 3.7, data yang diperlukan algoritma genetik untuk melakukan DNA SBH adalah panjang barisan DNA yang akan dicari (n), spektrum (S), probabilitas *crossover* (c), probabilitas mutasi (m), dan batas generasi. Secara umum, algoritma genetik untuk DNA *sequencing* adalah sebagai berikut:

begin
Baca data;
Tentukan parameter algoritma genetik ;
Bentuk populasi awal secara acak berukuran $N = \frac{1}{2} \times n$;
for generasi = 1 to batas generasi ***do***
Hitung nilai *fitness* tiap kromosom pada populasi;
Bentuk populasi orang tua dengan seleksi;
Lakukan *crossover* sebanyak *c* x *N;*
Lakukan mutasi sebanyak *m* x *N* x |*S*|;
Bentuk populasi baru untuk generasi selanjutnya;
***end for*;**
***end*;**

BAB IV

IMPLEMENTASI DAN BEBERAPA HASIL PERCOBAAN

4.1 IMPLEMENTASI

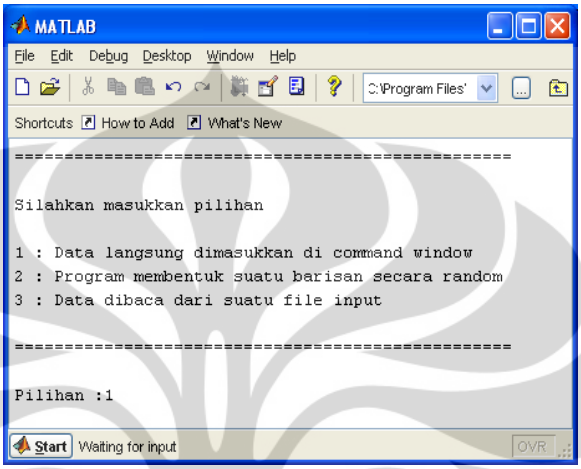
Pada Bab III telah dijelaskan DNA SBH dengan menggunakan algoritma genetik. Selanjutnya pada bab ini akan dijelaskan mengenai implementasi algoritma genetik tersebut menggunakan MATLAB 7.1. Program dijalankan pada *Personal Computer* dengan prosesor *Intel Pentium M*, memori 512 MB, dan sistem operasi *Microsoft Windows XP Professional*.

Berikut akan dijelaskan mengenai beberapa fungsi yang digunakan di dalam program. Algoritma genetik untuk DNA SBH secara keseluruhan akan dijalankan oleh fungsi SBH. Fungsi SBH akan memanggil fungsi POPULASI_AWAL, SELEKSI, CROSSOVER, MUTASI, dan POPULASI_BARU yang akan menjalankan tahap-tahap pada algoritma genetik yang meliputi pembentukan populasi awal, seleksi, *crossover*, mutasi, dan pembentukan populasi baru untuk generasi berikutnya. Penghitungan nilai *fitness* suatu kromosom akan dilakukan oleh fungsi FITNESS. Sedangkan untuk menghitung nilai *fits* yang digunakan pada proses *crossover*, akan dilakukan oleh fungsi FITS. Kemudian nilai *overlap*

degree antara dua fragmen yang bertetangga akan dihitung dengan menggunakan fungsi OVERLAPP. Karena algoritma genetik yang dibahas dalam skripsi ini hanya untuk DNA SBH dengan spektrum yang ideal, maka dalam implementasi program terlebih dahulu akan dicek apakah spektrum yang diberikan ideal atau tidak dengan menggunakan fungsi IDEAL. Untuk lebih jelas, *listing* program dapat dilihat pada Lampiran 1.

Pemanggilan program dilakukan dengan cara mengetik "SBH" pada MATLAB *Command Window*. Data yang dibutuhkan program adalah panjang barisan, spektrum, banyak fragmen anggota spektrum, panjang fragmen, probabilitas *crossover* (*c*), probabilitas mutasi (*m*), dan batas generasi. Pada implementasi program ini, pengguna diberikan tiga pilihan, yakni pilihan 1, 2, dan 3. Untuk pilihan 1, semua data yang dibutuhkan kecuali banyak fragmen anggota spektrum, dimasukkan langsung pada MATLAB *Command Window*. Untuk pilihan 2, pertama kali pengguna akan diminta untuk memasukkan panjang barisan DNA dan panjang fragmen yang diinginkan. Dari masukan tersebut, program akan membentuk barisan secara *random* dan spektrum dari barisan tersebut sesuai ukuran yang diinginkan pengguna. Kemudian program akan meminta pengguna memasukkan probabilitas *crossover*, probabilitas mutasi, dan batas generasi. Sedangkan untuk pilihan 3, semua data yang dibutuhkan kecuali probabilitas *crossover* (*c*), probabilitas mutasi (*m*), dan batas generasi akan disimpan dalam suatu *file input .* Format *file input* dapat dilihat pada Lampiran 2. Jadi, pada saat memanggil program,

tampilan yang pertama kali muncul pada *command window* adalah seperti Gambar 4.1



Gambar 4.1 Tampilan *command window* pilihan cara pemasukan data

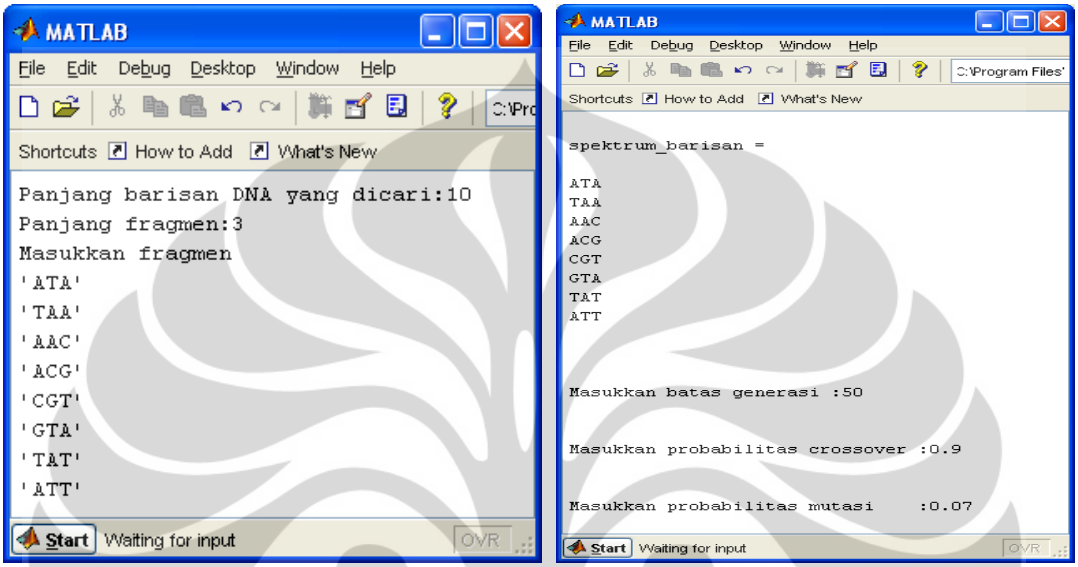
Program akan berhenti jika telah mencapai batas generasi yang telah ditentukan oleh pengguna. Kemudian program akan mengeluarkan kromosom terbaik, barisan DNA dengan panjang tertentu yang dibentuk oleh kromosom tersebut, dan grafik perubahan nilai *fitness* terbaik pada tiap generasi terhadap pertambahan generasi.

Contoh pemanggilan program untuk pilihan 1 akan diberikan pada Contoh 4.1.

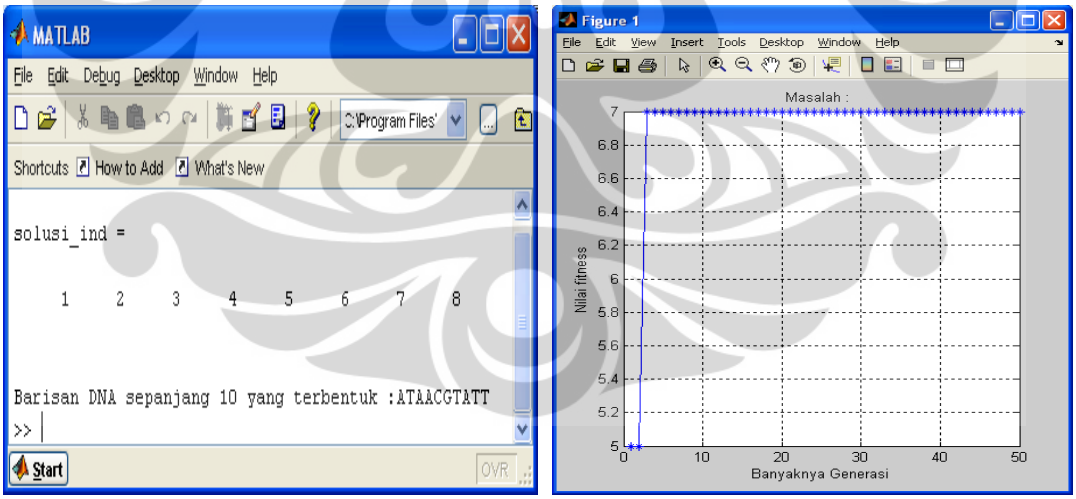
Contoh 4.1

Pada contoh ini, masalah pada Contoh 3.1 akan diselesaikan dengan menggunakan algoritma genetik. Data-data yang dibutuhkan seperti spektrum, panjang barisan DNA yang dicari, panjang fragmen, batas

generasi, probabilitas *crossover*, dan probabilitas mutasi akan diberikan langsung pada *command window.*



(a)



(b)

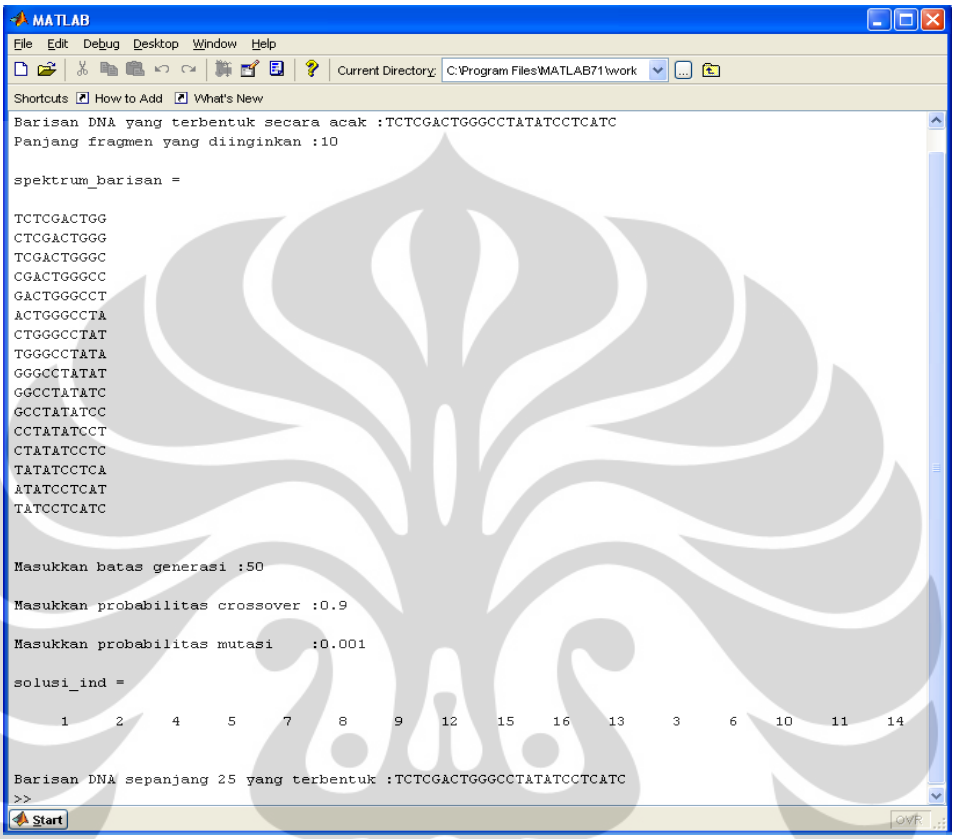
Gambar 4.2 Tampilan masukan (a) dan keluaran (b) untuk Contoh 4.1

Untuk pemanggilan program dengan pilihan 1, pengguna diminta untuk memberikan masukan berupa panjang barisan DNA yang dicari, panjang fragmen, spektrum, batas generasi, probabilitas *crossover*, dan probabilitas mutasi. Dalam masalah ini akan digunakan batas generasi = 50, c = 0.9, dan m = 0.07. Masukan dan keluaran untuk Contoh 3.1 diberikan pada Gambar 4.2.

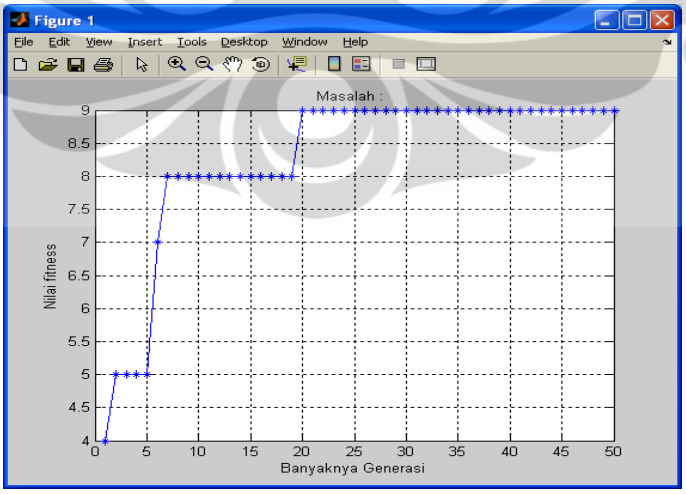
Dari Gambar 4.2 (b) terlihat bahwa solusi untuk Contoh 3.1 adalah kromosom dengan urutan fragmen 1-2-3-4-5-6-7-8 dengan barisan DNA sepanjang 10 yang terbentuk adalah *ATAACGTATT*. Hal ini berarti barisan DNA yang terbentuk sesuai dengan barisan DNA asal pada Contoh 3.1. Grafik pada Gambar 4.2 (b) menunjukkan nilai *fitness* terbaik pada tiap generasi. Dari grafik pada Gambar 4.2 (b) terlihat bahwa tidak terjadi perubahan nilai *fitness* terbaik pada selang generasi ke-3 sampai dengan generasi ke-50. Dapat dikatakan bahwa solusi yang diperoleh saat generasi ke-3 sudah optimal.

Selanjutnya, contoh pemanggilan program untuk pilihan 2 diberikan pada Contoh 4.2.

Contoh 4.2



(a)



(b)

Gambar 4.3 Tampilan *command window* (a) dan grafik perubahan nilai *fitness* (b) untuk contoh pemanggilan program dengan pilihan 2

Pada Gambar 4.3 (a), panjang barisan DNA dan panjang fragmen yang diinginkan pengguna secara berturut-turut adalah 25 dan 10. Nilai probabilitas *crossover* (*c*), probabilitas mutasi (*m*), dan batas generasi yang digunakan untuk Contoh 4.2 secara berturut-turut adalah 0.9, 0.001, dan 50. Barisan DNA yang terbentuk secara *random* adalah *TCTCGACTGGGCCTA TATCCTCATC* . Pada Gambar 4.3 (a) juga terlihat bahwa solusi yang diperoleh adalah barisan DNA *TCTCGACTGGGCCTAT ATCCTCATC.* Berarti solusi yang diperoleh sama dengan barisan DNA asal. Pada Gambar 4.3 (b) terlihat bahwa tidak terjadi perubahan nilai *fitness* selama selang generasi ke-20 sampai generasi ke-50. Oleh karena itu, dapat dikatakan bahwa untuk contoh pada Gambar 4.3 (a), solusi yang diperoleh sudah optimal sejak generasi ke-20.

Selanjutnya akan dibahas mengenai beberapa percobaan yang dilakukan untuk melihat kinerja algoritma genetik untuk beberapa masalah DNA SBH serta pengaruh parameter probabilitas *crossover* dan probabilitas mutasi terhadap kinerja algoritma genetik. Pada percobaan yang dilakukan, proses pemanggilan program menggunakan pilihan 3.

4.2 HASIL PERCOBAAN

Percobaan dilakukan untuk melihat kinerja algoritma genetik dalam melakukan DNA SBH dengan berbagai variasi ukuran masalah. Masalah

SBH yang digunakan untuk menguji algoritma dibagi menjadi dua jenis, yakni jenis 1 dan jenis 2. Jenis 1 merupakan masalah SBH yang diambil dari *Gen Bank* [DSS 09]. Sedangkan masalah SBH jenis 2 merupakan masalah yang dibuat dengan membentuk spektrum dari barisan DNA yang dibentuk secara acak. Masalah penguji jenis 1 terdiri dari 3 masalah DNA SBH yang masing-masing memiliki data sebagai berikut: barisan DNA yang akan dicari memiliki panjang 200, spektrum yang diberikan terdiri dari 191 fragmen dengan tiap fragmen memiliki panjang 10. Sedangkan masalah penguji jenis 2 terdiri dari tiga kelompok, yakni barisan DNA dengan panjang 25, 50, dan 75. Dari tiap barisan DNA tersebut, masing-masing dibentuk spektrum dengan panjang fragmen 6, 8, dan 10. Masalah penguji dinotasikan dengan sbh-*i* dimana *i* menyatakan panjang barisan DNA. Karena barisan DNA pada masalah penguji jenis 1 memiliki panjang yang sama, maka untuk membedakan masalah yang satu dengan masalah lainnya, pada notasi sbh-*i* ditambahkan alfabet a, b, dan c. Barisan DNA dari masalah penguji yang digunakan dapat dilihat pada Tabel 4.1a untuk masalah penguji jenis 1 dan Tabel 4.1b untuk jenis 2.

Masalah jenis 1 digunakan saat menguji keakuratan algoritma genetik untuk melakukan DNA *Sequencing by Hybridization* (DNA SBH) yang benar-benar telah dilakukan dalam kehidupan nyata seperti data yang terdapat pada *Gen Bank*. Sedangkan masalah penguji jenis 2 digunakan saat menguji

pengaruh perubahan parameter probabilitas *crossover* dan probabilitas mutasi terhadap kinerja algoritma genetik.

Tabel 4.1a Barisan DNA masalah penguji jenis 1

Masalah Penguji	Barisan DNA
sbh-200a	AGAATTGTGTGCTAAGTACTGGGGTAACACAGAGATGGTA ATTAATTCTTTCTGGAGAAACAGCAGTGAATGTGACAACCT GAGTGAGGGTGGAAAGCAGGAGAATTTCTAGACTCTGAAT TTCAGAAGCTTCCGGAACTAAAATGATGTGGAAGTTGAGT GAGTCTCGGTGGGATATTGCTGCTGCACCTGGTCAAGGC
sbh-200b	ATGGGAAATGTGGTCTCCGACGGATGGCCTACACCCGGG AGGAGTCCAGGAACTCTAGGTGGACAGTTACTTCCGCACA CGCGTAGTAGGACGGTAGCCGGTATTCAATCTTCAAATCA GCGCCGCGGGAAGTGCGGGCGGGTGTTGCTCCCCTGTC TCTGGACGACGCTGTGACTGATCCCAGGCTTGGAGCCGG AGCT
sbh-200c	TCTCATCCCCCTGGAAAGAAATTTCAAGGGATAAAGCACC ATGGATCTAACTTATATTCCCGAAGACCTATCCAGTTGTCC AAAATTTGTAAATAAATCCTGTCCTCCCACCAACCGCTCTT TTCATGTCCAGGTGATAATGTATTCGGTTATGACTGGAGC CATGATTATCACTATTTGGAAACTTGGTTATAATGGTT

Tabel 4.1b Barisan DNA masalah penguji jenis 2

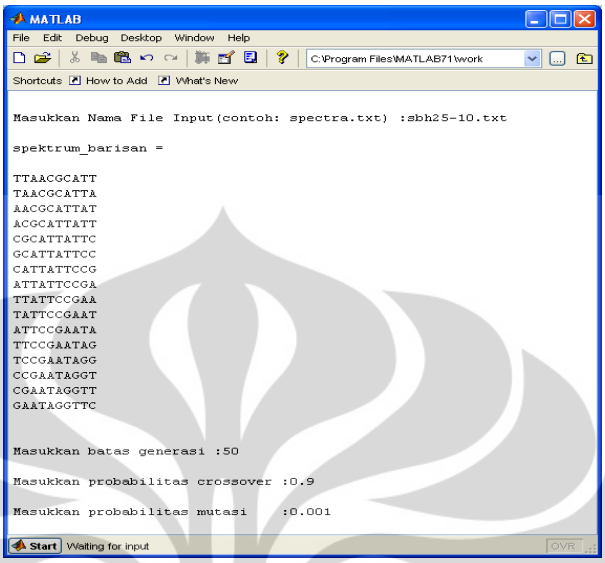
Masalah Penguji	Barisan DNA
sbh-25	TTAACGCATTATTCCGAATAGGTTC
sbh-50	ATGACTTTAAGATTTCCCATTGGAACCTGAAATTACGGATAG CCCCTTAG
sbh-75	AAAATGGTTTAAGTGGGAATTGGAGGGGTTGAGTATTAGAGT TATAGCCTGACCTGCCTAACCGCTATCGAACGG

Percobaan dilakukan dengan mengubah nilai parameter probabilitas *crossover* dan probabilitas mutasi. Sedangkan batas generasi untuk tiap

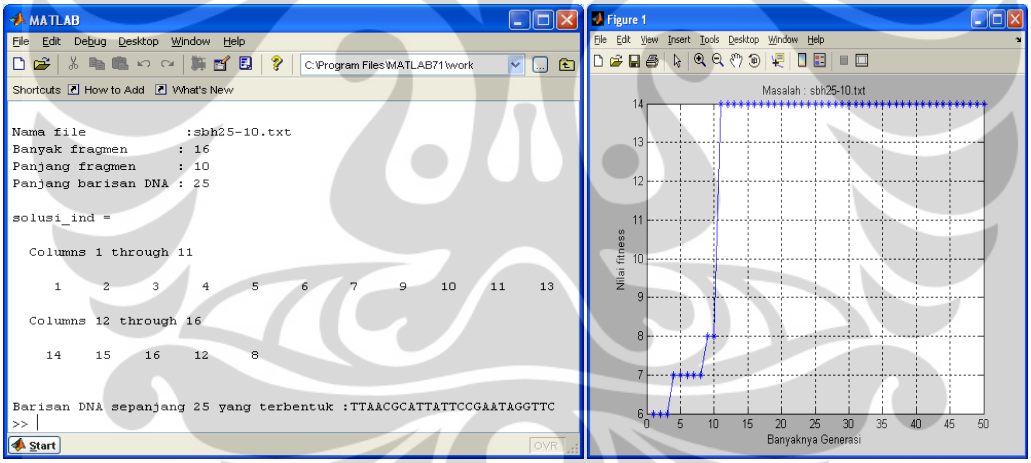
percobaan dibuat sama, yakni menggunakan batas generasi sampai generasi ke-50. Percobaan pertama menggunakan nilai probabilitas *crossover* (*c*) = 0.9 dan probabilitas mutasi (*m)* = 0.001, yang merupakan kombinasi nilai parameter terbaik untuk masalah DNA SBH menurut [JGKM 06]. Percobaan kedua menggunakan nilai probabilitas mutasi (*m)* = 0.001 dan nilai probabilitas *crossover* (*c*) berubah-ubah, selanjutnya percobaan ketiga menggunakan nilai probabilitas *crossover (c)* = 0.9 dan nilai probabilitas mutasi (*m*) berubah-ubah. Hal yang diperhatikan pada tiap percobaan adalah banyaknya solusi yang sesuai dengan barisan DNA asal yang diketahui.

Contoh tampilan *command window* saat pemanggilan program untuk percobaan yang dilakukan diberikan pada Gambar 4.4.

Pada Gambar 4.4 (a) terlihat bahwa *file* yang diuji adalah *file* sbh25-10.txt, yakni masalah penguji sbh-25 dengan fragmen anggota spektrum mempunyai panjang 10. Pada Gambar 4.4 (b) terlihat bahwa barisan DNA yang diperoleh adalah *TTAACGCATTATTCCGAATAGGTTC*. Berdasarkan Tabel 4.1 (b), barisan DNA untuk masalah penguji sbh-25 adalah *TTAACG CATTATTCCGAATAGGTTC.* Hal ini berarti solusi yang didapat pada Gambar 4.4 (b) sama dengan barisan DNA asal.



(a)



(b)

Gambar 4.4 Tampilan masukan (a) dan keluaran (b) untuk pemanggilan program dengan pilihan 3

Grafik pada Gambar 4.4 (b) menunjukkan bahwa pada selang generasi ke-11 sampai generasi ke-50, nilai f*itness* terbaik untuk tiap generasi bernilai tetap. Sebelumnya telah disebutkan bahwa solusi yang

diperoleh saat generasi ke-50 sama seperti barisan DNA asal. Hal ini berarti, solusi yang diperoleh sudah optimal sejak generasi ke-11.

Selanjutnya, hasil percobaan dengan masing-masing parameter akan diberikan pada Subbab 4.2.1 sampai 4.2.3.

4.2.1 Percobaan dengan Probabilitas *c* = 0.9 dan Probabilitas *m* = 0.001

Percobaan pertama dilakukan dengan generasi = 50, probabilitas *c* = 0.9, dan probabilitas *m* = 0.001. Masalah penguji yang digunakan pada percobaan pertama adalah masalah penguji jenis 1 dan 2. Untuk masalah penguji jenis 1, dibutuhkan waktu komputasi sekitar 8 jam untuk melakukan satu kali percobaan. Karena keterbatasan waktu yang ada, maka untuk masing-masing masalah penguji jenis 1 akan dilakukan percobaan sebanyak 5 kali. Sedangkan untuk masing-masing masalah penguji jenis 2, akan dilakukan percobaan sebanyak 10 kali.

Hasil percobaan diberikan pada Tabel 4.2a dan Tabel 4.2b, dimana kolom Masalah Penguji menyatakan masalah yang diuji, sedangkan kolom Banyak Solusi Optimal menyatakan banyaknya solusi yang sesuai dengan barisan DNA asal.

Tabel 4.2a Hasil percobaan untuk masalah penguji jenis 1 dengan *c* = 0.9, *m* = 0.001, dan generasi = 50 (5 kali percobaan)

Masalah Penguji	Panjang Fragmen	Banyak Fragmen	Ukuran Populasi	Banyak Solusi Optimal
sbh-200a	10	191	100	4
sbh-200b	10	191	100	3
sbh-200c	10	191	100	4

Tabel 4.2b Hasil percobaan untuk masalah penguji jenis 2 dengan *c* = 0.9, *m* = 0.001, dan generasi = 50 (10 kali percobaan)

Masalah Penguji	Panjang Fragmen	Banyak Fragmen	Ukuran populasi	Banyak Solusi Optimal
sbh-25	6	20	13	6
sbh-25	8	18	13	7
sbh-25	10	16	13	9
sbh-50	6	45	25	4
sbh-50	8	43	25	6
sbh-50	10	41	25	10
sbh-75	6	70	38	2
sbh-75	8	68	38	5
sbh-75	10	66	38	5

Berdasarkan hasil percobaan yang terlihat pada Tabel 4.2a, dalam 5 kali percobaan yang dilakukan, algoritma genetik memberikan hasil yang cukup baik. Untuk masalah penguji sbh-200a dan sbh-200c, dalam 5 kali percobaan didapat 4 solusi yang sama dengan barisan DNA asal. Sedangkan untuk masalah penguji sbh-200b, dalam 5 kali percobaan didapat 3 solusi yang sama dengan barisan DNA asal.

Selanjutnya dengan menggunakan nilai parameter dan panjang fragmen yang sama seperti pada data pada *Gen Bank* (c = 0.9, m = 0.001, generasi = 50, dan panjang fragmen = 10) akan dilihat apakah untuk barisan DNA yang lebih pende, algoritma genetik juga memberikan hasil yang baik atau tidak. Selain itu juga akan dilihat akibat yang dihasilkan jika panjang fragmen diperpendek. Hasil percobaan untuk masalah penguji sbh-25, sbh-50, dan sbh-75 diberikan pada Tabel 4.2b.

Berdasarkan data pada Tabel 4.2b, terlihat bahwa dengan menggunakan nilai parameter dan panjang fragmen yang sama dengan data pada *Gen Bank* (c = 0.9, m = 0.001, generasi = 50, dan panjang fragmen = 10), untuk barisan DNA dengan panjang 25, 50, dan 75, algoritma genetik juga memberikan hasil yang cukup baik. Untuk barisan DNA dengan panjang 25 dan 50, dalam 10 kali percobaan didapat solusi yang sama dengan barisan DNA asal, masing-masing sebanyak 9 dan 10. Sedangkan untuk barisan DNA dengan panjang 75, dalam 10 kali percobaan hanya didapat 5 solusi yang sama dengan barisan DNA asal. Pada hasil percobaan yang didapat juga terlihat bahwa untuk barisan DNA dengan panjang 25, 50, dan 75, semakin pendek fragmen maka solusi yang sama dengan barisan DNA asal yang diperoleh semakin sedikit.

Selanjutnya akan dilihat pengaruh perubahan nilai parameter c terhadap kinerja algoritma genetik.

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

Percobaan kedua dilakukan dengan menggunakan probabilitas *m* = 0.001 dan nilai probabilitas *c* yang berubah-ubah. Nilai *c* yang digunakan adalah 0.8, 0.85, 0.9, dan 0.95. Percobaan kedua dilakukan dengan tujuan untuk melihat pengaruh perubahan nilai parameter *c* terhadap kinerja algoritma genetik. Untuk tiap masalah penguji dilakukan percobaan sebanyak 10 kali.

Tabel 4.3a Hasil percobaan untuk masalah penguji jenis 2 dengan mengubah nilai parameter *c* (10 kali Percobaan)

Masalah Penguji	Panjang Fragmen	Banyak Fragmen	Banyak Solusi Optimal untuk Tiap Probabilitas *Crossover*			
			0.8	0.85	0.9	0.95
sbh-25	6	20	4	5	6	6
sbh-25	8	18	5	7	7	9
sbh-25	10	16	7	8	9	9
sbh-50	6	45	1	2	4	4
sbh-50	8	43	1	5	6	7
sbh-50	10	41	4	10	10	7
sbh-75	6	70	1	2	2	3
sbh-75	8	68	1	4	5	5
sbh-75	10	66	3	5	5	6

Pada Tabel 4.3a terlihat bahwa penambahan nilai probabilitas *crossover* mengakibatkan banyak solusi optimal yang didapat cenderung menjadi semakin banyak, seperti yang terjadi pada masalah penguji sbh-25

dengan semua panjang fragmen yang dicoba, sbh-50 dengan panjang fragmen 6 dan 8, dan sbh-75 dengan semua panjang fragmen. Untuk masalah penguji sbh-50 dengan panjang fragmen 10 ternyata perubahan nilai probabilitas *crossover* dari 0.9 menjadi 0.95 justru menyebabkan banyaknya solusi optimal yang diperoleh menjadi berkurang. Hal tersebut kemungkinan terjadi karena pengaruh faktor *random* pada algoritma genetik.

Tabel 4.3b Banyak proses *crossover* yang terjadi untuk masalah penguji jenis 2 dengan mengubah nilai parameter *c*

Masalah Penguji	Panjang Fragmen	Banyak Fragmen	Banyak *Crossover* untuk Tiap Probabilitas *Crossover*			
			0.8	0.85	0.9	0.95
sbh-25	6	20	10	11	12	12
sbh-25	8	18	10	11	12	12
sbh-25	10	16	10	11	12	12
sbh-50	6	45	20	21	23	24
sbh-50	8	43	20	21	23	24
sbh-50	10	41	20	21	23	24
sbh-75	6	70	30	32	34	36
sbh-75	8	68	30	32	34	36
sbh-75	10	66	30	32	34	36

Berdasarkan penjelasan pada Bab III, proses *crossover* yang terjadi adalah sebanyak $c \times N$, dengan N menyatakan ukuran populasi. Banyaknya proses *crossover* yang terjadi untuk masing-masing nilai probabilitas *crossover* dapat dilihat pada Tabel 4.3b. Pada Tabel 4.3b terlihat bahwa penambahan nilai probabilitas *crossover* sebanyak 0.5 akan menambah banyaknya prosess *crossover* yang terjadi untuk masalah penguji sbh-25, sbh-50, dan sbh-75 sebanyak 1 atau 2.

Pada percobaan berikutnya, akan diuji pengaruh perubahan nilai probabilitas mutasi terhadap kinerja algoritma genetik.

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

Percobaan ketiga dilakukan untuk melihat pengaruh perubahan nilai parameter probabilitas mutasi (*m*) terhadap kinerja algoritma genetik. Percobaan ketiga dilakukan dengan melakukan perubahan pada nilai probabilitas mutasi. Probabilitas mutasi yang digunakan yakni 0.001, 0.005, 0.01, dan 0.015. Seperti pada percobaan kedua, masalah penguji yang digunakan adalah masalah penguji jenis 2. Percobaan dilakukan sebanyak 10 kali untuk masing-masing masalah penguji. Hasil percobaan diberikan pada Tabel 4.4a.

Berdasarkan hasil percobaan pada Tabel 4.4a, terlihat bahwa untuk masalah penguji sbh-25 dengan semua panjang fragmen yang digunakan, pertambahan nilai probabilitas mutasi menyebabkan semakin banyak solusi optimal yang diperoleh. Hal yang sama terjadi untuk masalah penguji sbh-75. Namun, hasil yang diperoleh pada masalah penguji sbh-75 tidak sebaik pada masalah penguji sbh-25. Pada masalah penguji sbh-25 dengan semua panjang fragmen, penambahan nilai probabilitas mutasi hingga bernilai 0.01 dan 0.015 menyebabkan banyak solusi optimal yang diperoleh sama dengan

banyaknya percobaan yang dilakukan. Untuk masalah penguji sbh-50 dengan panjang fragmen 10, penambahan nilai probabilitas dari 0.005 menjadi 0.01 justru mengurangi banyak solusi optimal yang diperoleh. Seperti yang terjadi pada percobaan sebelumnya, hal tersebut kemungkinan disebabkan oleh faktor *random* yang terjadi pada algoritma genetik.

Tabel 4.4a Hasil percobaan untuk masalah penguji jenis 2 dengan mengubah nilai parameter *m* (10 kali Percobaan)

Masalah Penguji	Panjang Fragmen	Banyak Fragmen	Banyak Solusi Optimal untuk Tiap Probabilitas Mutasi			
			0.001	0.005	0.01	0.015
sbh-25	6	20	6	7	10	10
sbh-25	8	18	7	7	10	10
sbh-25	10	16	9	9	10	10
sbh-50	6	45	4	4	3	4
sbh-50	8	43	6	5	6	6
sbh-50	10	41	10	10	6	6
sbh-75	6	70	2	3	5	5
sbh-75	8	68	5	5	6	6
sbh-75	10	66	5	5	7	7

Proses mutasi yang terjadi adalah sebanyak $m \times N \times |S|$, dengan $|S|$ menyatakan banyaknya fragmen dan N menyatakan ukuran populasi. Banyaknya proses mutasi yang terjadi untuk masing-masing nilai probabilitas mutasi dapat dilihat pada Tabel 4.4b.

Tabel 4.4b Banyaknya proses mutasi yang terjadi untuk masalah penguji jenis 2 dengan mengubah nilai parameter *m*

Masalah Penguji	Panjang Fragmen	Banyak Fragmen	Banyak Mutasi untuk Tiap Probabilitas Mutasi			
			0.001	0.005	0.01	0.015
sbh-25	6	20	0	1	3	4
sbh-25	8	18	0	1	2	4
sbh-25	10	16	0	1	2	3
sbh-50	6	45	1	6	11	17
sbh-50	8	43	1	5	11	16
sbh-50	10	41	1	5	10	15
sbh-75	6	70	3	13	27	40
sbh-75	8	68	3	13	26	39
sbh-75	10	66	3	13	25	38

Pada Tabel 4.4b terlihat bahwa untuk masalah penguji sbh-50 dan sbh-75, pertambahan sedikit nilai probabilitas mutasi mengakibatkan pertambahan yang cukup signifikan pada banyaknya proses mutasi yang terjadi. Pada masalah penguji sbh-25, saat nilai probabilitas mutasi 0.001, pada masalah penguji tersebut tidak terjadi proses mutasi. Namun, dengan meningkatkan nilai probabilitas mutasi menjadi 0.005, 0.01, dan 0.015 mengakibatkan terjadinya proses mutasi pada masalah penguji tersebut.

Terlihat bahwa penambahan nilai probabilitas mutasi memberikan pengaruh yang lebih signifikan dibandingkan dengan pengaruh yang disebabkan oleh penambahan nilai probabilitas *crossover*. Hal tersebut karena terjadi pertambahan yang signifikan pada banyaknya proses mutasi yang terjadi jika probabilitas mutasi dinaikkan, sehingga kemungkinan

populasi akan terjebak pada solusi optimal lokal semakin kecil. Misalnya saja pada masalah penguji sbh-25. Pada Tabel 4.3a (generasi = 50 dan *m* = 0.001) terlihat bahwa saat nilai probabilitas *crossover* dinaikkan dari 0.9 menjadi 0.95, untuk sbh-25 dengan panjang fragmen 8 dan 10, tidak terjadi pertambahan pada banyaknya solusi optimal yang diperoleh. Sedangkan pada sbh-25 dengan panjang fragmen 6, pertambahan nilai probabilitas *crossover* tersebut hanya menambah banyaknya solusi optimal yang diperoleh sebanyak 2. Namun demikian, Pada Tabel 4.4a (generasi = 50 dan *c* = 0.9) terlihat bahwa jika probabilitas *crossover* dibuat tetap bernilai 0.9, dan nilai probabilitas mutasi dinaikkan menjadi 0.01 dan 0.015, dalam percobaan yang dilakukan, semua solusi yang diperoleh sama seperti barisan DNA asal.

BAB V

KESIMPULAN

Berdasarkan pembahasan dan hasil percobaan pada Bab IV, dapat dikatakan bahwa algoritma genetik cukup baik digunakan pada DNA *sequencing by hybridization* untuk tiga masalah yang diambil dari *Gen Bank* dengan barisan DNA pada masing-masing masalah mempunyai panjang 200 dan panjang fragmen anggota spektrum adalah 10. Untuk dua masalah yang diambil dari *Gen Bank*, dari 5 kali percobaan yang dilakukan dengan menggunakan probabilitas mutasi m = 0.001, probabilitas *crossover* c = 0.9, dan generasi = 50, ternyata didapat solusi yang sesuai dengan barisan DNA asal sebanyak 4. Sedangkan untuk masalah lainnya, dalam 5 kali percobaan didapat solusi yang sesuai sebanyak 3. Selain itu, hasil yang cukup baik juga didapat saat percobaan tersebut (m = 0.001, c = 0.9, generasi =50, dan panjang fragmen = 10) dilakukan dengan menggunakan barisan DNA dengan panjang 25, 50, dan 75. Untuk barisan DNA dengan panjang 25, 50, dan 75, dari 10 kali percobaan yang dilakukan, solusi yang sesuai dengan barisan DNA asal yang diperoleh tidak kurang dari 5. Untuk masalah penguji sbh-25, sbh-50, dan sbh-75, ternyata jika digunakan panjang fragmen 6 dan 8,

banyaknya solusi yang sesuai dengan barisan DNA asal menjadi lebih sedikit.

Selain itu, hasil percobaan yang dilakukan juga menunjukkan bahwa nilai probabilitas *crossover* dan probabilitas mutasi berpengaruh pada kinerja algoritma genetik dalam memperoleh solusi. Dari hasil percobaan yang diperoleh terlihat bahwa ketika nilai probabilitas *crossover* dinaikkan dari 0.8 menjadi 0.85, 0.9, dan 0.95, solusi yang diperoleh semakin mirip dengan barisan DNA asal, yang ditandai dengan semakin banyak solusi yang sama dengan barisan DNA asal. Hal yang sama juga terjadi ketika nilai probabilitas mutasi dinaikkan dari 0.001 menjadi 0.005, 0.01, dan 0.015.

Berdasarkan hasil percobaan juga terlihat bahwa untuk masalah sbh-25, peningkatan nilai probabilitas mutasi memberikan pengaruh yang lebih signifikan dibandingkan dengan peningkatan nilai probabilitas *crossover*. Hal tersebut dikarenakan saat nilai probabilitas mutasi sangat kecil, yakni m = 0.001, pada sbh-25 tidak terjadi proses mutasi. Sedangkan saat nilai probabilitas mutasi diperbesar (pada percobaan 4.23) pada sbh-25 terjadi proses mutasi sehingga kemungkinan populasi terjebak dalam solusi optimal lokal menjadi lebih kecil.

Algorima genetik banyak dipengaruhi oleh faktor *random*, Oleh sebab itu, penambahan nilai parameter probabilitas *crossover* dan probabilitas mutasi terkadang justru tidak memperbaiki kinerja algoritma genetik dalam memperoleh solusi.

DAFTAR PUSTAKA

[BK 06] Blazewicz, Jacek dan Kasprzak, Marta. Computational complexity of isothermic DNA sequencing by hybridization. *Discrete Applied Mathematics* 154 (2006) 718 – 729.

[DS 09] *DNA Sequencing*. http://genome.wellcome.ac.uk/Resources/Technology-centre/Dna-sequencing, 5 Agustus 2009, pk. 21.30.

[GA 09] *Genetic Algorithm*. http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/tcw2/report.html. 27 Oktober 2009, 27 Oktober 2009, pk.12.10.

[IGA 09] *Introduction to Genetic Algorithm*. http://lancet.mit.edu/~mbwall/presentations/IntroToGAs/, 27 Oktober 2009, pk. 12.20.

[JGKM 06] Jacek, B. , Glover, F. , Kasprzak, M. , Markiewicz, W.T. Algoritm Dealing with repetitions in sequencing by hybridization. *Computational Biology and Chemistry* 30 (2006) 313 –320. (www.elsevier.com/locate/combiolchem)

[Kas 04] Kasprzak, M. An algorithm for isothermic DNA sequencing. *Bulletin of The Polish Academy of Sciences Technical Sciences* Vol. 52, No. 1, 2004.

[MA 09] *Metaheuristic Algoritm*. www.lapetoule.com/meta/en/metaheuristic_en.pdf , 5 Agustus 2009, pk. 21.00.

[SA 04] Sarma, A. K. dan Ahmed, J. A. 2004. *Optimal Operating Policy for a Multipurpose Reservoir using Genetic Algorithm*. www.actapress.com/PDFViewer.aspx?paperId=16157, 27 Oktober 2009, pk. 12.40.

[Wik 09] Wikipedia.http://www.wikipedia.org, 5 Agustus 2009, pk. 22.30.

[DSS 09] Download Standard Spectra. http://bio.cs.put.poznan.pl/index.php/research/36-current-research/64-download-spectra, 27 Oktober 2009. pk. 14.00.

LAMPIRAN 1

Listing Program Algoritma Genetik

- **SBH**

```
%ALGORITMA GENETIK UNTUK DNA SEQUENCING BY HYBRIDIZATION

clc;
clear;

fprintf('================================================\n\n');
fprintf('Silahkan masukkan pilihan\n\n');
fprintf('1 : Data langsung dimasukkan di command window\n');
fprintf('2 : Program membentuk suatu barisan secara random\n');
fprintf('3 : Data dibaca dari suatu file input\n\n');
fprintf('================================================\n\n');

pilihan=input('Pilihan :');
clc;

%baca data
[problem,spektrum,generasi,Pc,Pm,panjang_sequence,...
                  banyak_oligo, panjang_fragmen] = INTRO(pilihan);

if IDEAL(spektrum)==0
    fprintf('SPEKTRUM tidak ideal\n');
    fprintf('MAAF program hanya untuk spektrum ideal');
    break;
end

%membentuk populasi awal
[populasi]=POPULASI_AWAL(panjang_sequence,banyak_oligo);

iterasi=0;

while iterasi < generasi

   iterasi=iterasi+1;

   %menghitung nilai fitness masing-masing individu dalam populasi
         [nilai_fitness]=HITUNG_FITNESS(populasi,spektrum,...
                              panjang_fragmen,panjang_sequence);

   %melakukan proses seleksi
   [parent]=SELEKSI(populasi,nilai_fitness);
```

```
   %melakukan proses crossover
   [child]=CROSSOVER(parent,banyak_oligo,panjang_fragmen,...
                     spektrum,Pc);

   %melakukan proses mutasi
   [child,parent]=MUTASI(banyak_oligo,panjang_fragmen,...
                         parent,child,spektrum,Pm);

   %membentuk populasi baru
   [populasi,fitness_best]=POPULASI_BARU(parent,child,spektrum,...
                                 panjang_fragmen,panjang_sequence);

   plot_f(iterasi,:)=[fitness_best];

end

%hasil akhir
SOLUSI(populasi,spektrum,panjang_fragmen,panjang_sequence)

%membuat plot
plot(plot_f,'-*');
title(['Masalah : ',problem])
xlabel('Banyaknya Generasi')
ylabel('Nilai fitness')
grid on
```

- **INTRO**

```
function[problem,spektrum,generasi,Pc,Pm,...
         panjang_sequence,banyak_oligo,panjang_fragmen] = INTRO(z)

if z==1
    problem=0;
    panjang_sequence=input('Panjang barisan DNA yang dicari:');
    panjang_fragmen=input('Panjang fragmen:');
    fprintf('Masukkan fragmen\n');
    for i=1:panjang_sequence-panjang_fragmen+1
        matriks(i,1:panjang_fragmen)=input('');
    end
    clc;
    spektrum=matriks'
    panjang_fragmen=size(spektrum,1);
    banyak_oligo=size(spektrum,2);
    [generasi,Pc,Pm]=INITIALISASI(1);
    clc;
end
```

```
if z==2
    problem=0;
    panjang_sequence=input('Panjang barisan DNA yang dicari:');
    for i=1:panjang_sequence
        barisan(i,:)=round(1+3*rand);
    end
    fprintf('Barisan DNA yang terbentuk secara acak :');
    for j=1:panjang_sequence
        if barisan(j,:)==1
            fprintf('A');
        end
        if barisan(j,:)==2
            fprintf('T');
        end
        if barisan(j,:)==3
            fprintf('C');
        end
        if barisan(j,:)==4
            fprintf('G');
        end
    end
    fprintf('\n');
    panjang_fragmen=input('Panjang fragmen yang diinginkan :');
    spektrum=BENTUK_SPEKTRUM(barisan,panjang_fragmen,...
                             panjang_sequence)
    while IDEAL(spektrum)==0
        fprintf('Spektrum yang terbentuk tidak IDEAL\n');
        fprintf('Masukkan panjang fragmen yang lain\n');
        panjang_fragmen=input('Panjang fragmen yang diinginkan :');
        spektrum=BENTUK_SPEKTRUM(panjang_fragmen,panjang_sequence)
    end
    banyak_oligo=size(spektrum,2);
    [generasi,Pc,Pm]=INITIALISASI(1);
end

if z==3

    problem = input('\nMasukkan Nama File Input(contoh: spectra.txt)
    :','s');
    matriks = fopen(problem,'r');
    while matriks<0
      problem = input('\n\nNama file salah, silahkan ulangi (contoh:
      spectra.txt) :','s');
      matriks = fopen(problem,'r');
    end

    %panjang barisan DNA yang akan dicari
    panjang_sequence=fscanf(matriks,'%d',[1,1]);


    %banyak fragmen dan panjang tiap fragmen
```

```
    N =(fscanf(matriks,'%d',[1,2]));

    banyak_oligo = N(1);
    panjang_fragmen = N(2);

    %fragmen
    spektrum = (fscanf(matriks,'%s',[N(2),N(1)]));
    spektrum

    fclose(matriks);

    [generasi,Pc,Pm]=INITIALISASI(1);
    clc;

    fprintf('\nNama file            :%s\n',problem)
    fprintf('Banyak fragmen     : %d\nPanjang fragmen       :
    %d\nPanjang barisan DNA   : %d\n',N(1),N(2),panjang_sequence)

end
```

- **INITIALISASI**

```
function[generasi,Pc,Pm] = INITIALISASI(z)

if z==1

      %Inisialisasi parameter
      generasi = input('\n\nMasukkan batas generasi   :');

      Pc = input('\n\nMasukkan probabilitas crossover :');

      Pm = input('\n\nMasukkan probabilitas mutasi    :');

end
```

- **POPULASI_AWAL**

```
function[populasi]=POPULASI_AWAL(panjang_sequence,banyak_oligo)

%ukuran populasi
p=round(panjang_sequence/2);

%membentuk permutasi secara acak dari bilangan 1,2,…,|S| sebanyak
ukuran populasi
```

```
for i=1:p
   sementara(i,:)=randperm(banyak_oligo);
end

populasi=sementara';
```

- **FITNESS**

```
function[nilai_fitness]=FITNESS(individu,spektrum,...
                            panjang_fragmen,panjang_sequence)

nilai_fitness=0;
a=size(individu,1);
uk_vek=1;

oligo=1;
panjang=panjang_fragmen;

while oligo<=a
   for j=oligo+1:a
      b=OVERLAPP(individu(j-1,:),individu(j,:),spektrum,...
                 panjang_fragmen);
      panjang=panjang+panjang_fragmen-b;
      if panjang>panjang_sequence
         vek_fitness(uk_vek,:)=j-oligo;
         uk_vek=uk_vek+1;
         break;
      end
   end
   if panjang<=panjang_sequence
      vek_fitness(uk_vek,:)=j-oligo+1;
      uk_vek=uk_vek+1;
      break;
   end
   oligo=oligo+1;
end
nilai_fitness=max(vek_fitness);
```

- **HITUNG_FITNESS**

```
function[nilai_fitness]=HITUNG_FITNESS(populasi,spektrum,...
                            panjang_fragmen,panjang_sequence)

sizepop=size(populasi,2);
```

```
for i=1:sizepop
   nilai_fitness(i,:)=FITNESS(populasi(:,i),spektrum,...
                       panjang_fragmen,panjang_sequence);
end
```

- **SELEKSI**

```
function[parent]=SELEKSI(populasi,nilai_fitness)

F=nilai_fitness;

p=size(populasi,2);

%menghitung nilai fitness rata-rata
jumlah=0;
for i=1:p
   jumlah=jumlah+F(i,:);
end

F_rata=jumlah/p;

%menghitung nilai fp untuk masing-masing kromosom
for j=1:p
   if j==1
      fp(j,:)=F(j,:)/F_rata;
   else
      fp(j,:)=(F(j,:)/F_rata)+fp(j-1,:);
   end
end

%menentukan kromosom yang akan menjadi anggota populasi orang tua
r=rand;
count=1;
k=1;
m=1;
while count<=p
   if r<fp(k,:)
      parent(:,m)=populasi(:,k);
      r=r+1;
      count=count+1;
      m=m+1;
   else
      k=k+1;
   end
end
```

- **CROSSOVER**

```
function[child]=CROSSOVER(parent,banyak_oligo,panjang_fragmen,...
                          spektrum,Pc)

banyak_parent=size(parent,2);

%menghitung banyaknya proses crossover yang dilakukan
r1=round(Pc*banyak_parent);
child=zeros(banyak_oligo,r1);

for i=1:r1

   %memilih pasangan orang tua
   acak1=round(1+(banyak_parent-1)*rand);
   acak2=round(1+(banyak_parent-1)*rand);
   while acak1==acak2
      acak2=round(1+(banyak_parent-1)*rand);
   end
   parent1=parent(:,acak1);
   parent2=parent(:,acak2);

   %memilih fragmen yangg pertama ada di blok kromosom anak
   r2=round(1+(banyak_oligo-1)*rand);
    C=[r2];
   oligo=1;

   while oligo<banyak_oligo
      a=size(C,1);
      %menghitung nilai fit dari masing-masing predecessor

      fit_pred=zeros(2,1);
      vektor_pred=zeros(2,1);
      pred=1;
      for j=1:banyak_oligo
         if j~=banyak_oligo
            if parent1(j+1,:)==C(1,:)
               pilih11=parent1(j,:);
               fit_pred(pred,:)=FIT([pilih11;C],spektrum,...
                                    panjang_fragmen);
               vektor_pred(pred,:)=pilih11;
               pred=pred+1;
            end
            if parent2(j+1,:)==C(1,:)
               pilih12=parent2(j,:);
               fit_pred(pred,:)=FIT([pilih12;C],spektrum,...
                                    panjang_fragmen);
               vektor_pred(pred,:)=pilih12;
               pred=pred+1
            end
```

```
   end
end
sum_fit_pred=[fit_pred vektor_pred];

%menghitung nilai fit dari masing-masing successor
fit_suc=zeros(2,1);
vektor_suc=zeros(2,1);
suc=1;
for k=1:banyak_oligo
   if k~=1
      if parent1(k-1,:)==C(a,:)
         pilih21=parent1(k,:);
         fit_suc(suc,:)=FIT([C;pilih21],spektrum,...
                            panjang_fragmen);
         vektor_suc(suc,:)=pilih21;
         suc=suc+1;
      end
      if parent2(k-1,:)==C(a,:)
         pilih22=parent2(k,:);
         fit_suc(suc,:)=FIT([C;pilih22],spektrum,...
                            panjang_fragmen);
         vektor_suc(suc,:)=pilih22;
         suc=suc+1;
      end
   end
end
sum_fit_suc=[fit_suc vektor_suc];

%menentukan fragmen(predecessor atau successor)
%yang akan menjadi bagian dari kromosom anak
for m=1:2
      %jika predecessor sudah ada dalam kromosom anak
      if CEK(C,sum_fit_pred(m,2))==0
            sum_fit_pred(m,1)=0;
            sum_fit_pred(m,2)=0;
      end
      %jika successor sudah ada dalam kromosom anak
      if CEK(C,sum_fit_suc(m,2))==0
            sum_fit_suc(m,1)=0;
            sum_fit_suc(m,2)=0;
      end
end

sum_fit_pred=sortrows(sum_fit_pred);
sum_fit_suc=sortrows(sum_fit_suc);
max1=sum_fit_pred(2,1); % nilai fit maksimum dari predecessor
max2=sum_fit_suc(2,1);  % nilai fit maksimum dari successor

%proses yang dilakukan jika tidak ada predecessor
%atau successor yang memenuhi
if sum_fit_pred(:,2)==zeros(2,1)&sum_fit_suc(:,2)==zeros(2,1)
```

```
            C=CROSSOVER_2(C,parent1,spektrum,panjang_fragmen,...
                 banyak_oligo);
        else
           %proses yang dilakukan jika ada predecessor
           %atau successor yang memenuhi
           if max1<max2
              if sum_fit_suc(1,1)==sum_fit_suc(2,1)
                 r3=round(1+rand);
                 anak=[C;sum_fit_suc(r3,2)];
              else
                  anak=[C;sum_fit_suc(2,2)];
              end
              C=anak;
           else
              if max1>max2
                 if sum_fit_pred(1,1)==sum_fit_pred(2,1)
                    r4=round(1+rand);
                    anak=[sum_fit_pred(r4,2);C];
                 else
                     anak=[sum_fit_pred(2,2);C];
                 end
                 C=anak;
              else
                 pilih=[sum_fit_pred(:,2);sum_fit_suc(:,2)];
                 r5=round(1+3*rand);
                 while pilih(r5,:)==0
                    r5=round(1+3*rand);
                 end

                 if r5==1|r5==2
                    anak=[pilih(r5,:);C];
                 else
                    anak=[C;pilih(r5,:)];
                 end
                 C=anak;
              end
           end
        end

        oligo=oligo+1;
     end
     child(:,i)=C;
end
```

- **CROSSOVER_2**

```
function[vektor]=CROSSOVER_2(C,parent1,spektrum,...
                        panjang_fragmen,banyak_oligo)

j=1;
uk_C=size(C,1);

%menghitung nilai overlap antara fragmen yang belum terdapat
%pada blok kromosom anak dengan fragmen urutan pertama atau urutan
terakhir
%pada blok kromosom anak
%baris j untuk predecessor dan j+1 untuk successor

for i=1:banyak_oligo
   if CEK(C,parent1(i,:))==1

jumlah_overlapp(j,:)=OVERLAPP(parent1(i,:),C(1,:),spektrum,panjang_f
ragmen);

jumlah_overlapp(j+1,:)=OVERLAPP(C(uk_C,:),parent1(i,:),spektrum,panj
ang_fragmen);
      list_oligo(j,:)=parent1(i,:);
      list_oligo(j+1,:)=parent1(i,:);
      j=j+1;
   end
end

m=1;
uk_jumlah_overlapp=size(jumlah_overlapp,1);

%memilih calon fragmen yang akan ditukar posisinya
for k=1:uk_jumlah_overlapp
   if jumlah_overlapp(k,:)==max(jumlah_overlapp)
      pilih_oligo(m,:)=k;
      m=m+1;
   end
end

uk_pilih_oligo=size(pilih_oligo,1);
r=round(1+(uk_pilih_oligo-1)*rand);

%kalau genap berarti successor yang terpilih
%kalau ganjil berarti predecessor yang terpilih

if mod(pilih_oligo(r,:),2)==0
   vektor=[C;list_oligo(pilih_oligo(r,:),:)];
else
   vektor=[list_oligo(pilih_oligo(r,:),:);C];
end
```

- **MUTASI**

```
function[child,parent]=MUTASI(banyak_oligo,panjang_fragmen,parent,..
                              child,spektrum,Pm)

A=child;
P=parent;
B=[P A];
m=banyak_oligo;
P1=size(P,2);
B1=size(B,2);

banyak_mutasi=round(Pm*P1*banyak_oligo);

for mut=1:banyak_mutasi

%menentukan individu yang akan mengalami mutasi
r=round(1+(B1-1)*rand);

    %menentukan nilai tod(pred oi,oi,suc oi)tiap fragmen oi
    for i=1:m
      if i==1
         M(i)=OVERLAPP(B(i,r),B(i+1,r),spektrum,panjang_fragmen);
      else
         if i==m
            M(i)=OVERLAPP(B(i-1,r),B(i,r),spektrum,panjang_fragmen);
         else
            M(i)=OVERLAPP(B(i-1,r),B(i,r),spektrum,panjang_fragmen)
                +OVERLAPP(B(i,r),B(i+1,r),spektrum,panjang_fragmen);
         end
      end
    end

    %menentukan fragmen dengan nilai tod(pred oi,oi,suc oi) minimum

    for k=1:m
      if M(k)==min(M)
        break;
      end
    end

    %melakukan penukaran posisi fragmen
    %jika fragmen yang terpilih merupakan
    %fragmen dengan urutan pertama dalam individu

    if k==1
        sementara1=B(1,r);
        B(1,r)=B(m,r);
        B(m,r)=sementara1;
    else
```

```
        %jika fragmen yang terpilih merupakan
        %fragmen dengan urutan terakhir

        if k==m
            sementara1=B(m,r);
            B(m,r)=B(1,r);
            B(1,r)=sementara1;
        else
            %jika nilai od fragmen dengan predecessornya lebih kecil
            %daripada dengan successornya

            if OVERLAPP(B(k-1,r),B(k,r),spektrum,panjang_fragmen)
               <OVERLAPP(B(k,r),B(k+1,r),spektrum,panjang_fragmen)

                sementara2=B(k,r);
                B(k,r)=B(k-1,r);
                B(k-1,r)=sementara2;
            else
                %jika nilai od fragmen dengan successornya lebih
                %kecil
                %daripada dengan predecessornya

               if OVERLAPP(B(k-1,r),B(k,r),spektrum,panjang_fragmen)
                 >OVERLAPP(B(k,r),B(k+1,r),spektrum,panjang_fragmen)

                    sementara2=B(k,r);
                    B(k,r)=B(k+1,r);
                    B(k+1,r)=sementara2;
                else
                    %jika nilai od fragmen dengan predecessor dan
                    %successornya sama
                    q=round(rand);
                    %menyatakan predecessor yg terpilih
                    if q==1
                        sementara2=B(k,r);
                        B(k,r)=B(k-1,r);
                        B(k-1,r)=sementara2;
                    else
                        %menyatakan successor yg terpilih
                        sementara2=B(k,r);
                        B(k,r)=B(k+1,r);
                        B(k+1,r)=sementara2;
                    end
                end
            end
        end
    end
    child=B(:,P1+1:B1);
    parent=B(:,1:P1);
end
```

- **CEK**

```
function[nilai]=CEK(kromosom,oligo)

p=size(kromosom,1);

nilai=1;
for i=1:p
   if kromosom(i,:)==oligo
      nilai=0;
      break;
   end
end
```

- **FITS**

```
function[nilai_fit]=FITS(individu,spektrum,panjang_fragmen)

num_fragmen=size(individu,1);
nilai_fit=num_fragmen/PANJANG(individu,spektrum,panjang_fragmen);
```

- **OVERLAPP**

```
function[nilai_od]=OVERLAPP(oligo1,oligo2,spektrum,panjang_fragmen)

N=spektrum;
r=oligo1;
s=oligo2;
k=1;
nilai_od=0;
i=1;
while i<=panjang_fragmen
   if N(i,r)==N(k,s)
      nilai_od=nilai_od+1;
      i=i+1;
      k=k+1;
   else
      nilai_od=0;
      if k==1
         i=i+1;
      end
      k=1;
   end
end
```

- **PANJANG**

```
function[panjang]=PANJANG(individu,spektrum,panjang_fragmen)

seq_ind=SEQUENCE(individu,spektrum,panjang_fragmen);
panjang=size(seq_ind,1);
```

- **POPULASI_BARU**

```
function[populasi,fitness_best]=POPULASI_BARU(parent,child,...
        spektrum,panjang_fragmen,panjang_sequence,banyak_oligo)

sizeparent=size(parent,2);
sizechild=size(child,2);

%menghitung nilai fitness parent
for i=1:sizeparent

vektor_fitness(i,:)=FITNESS(parent(:,i),spektrum,panjang_fragmen,pan
jang_sequence);
   vektor_parent(i,:)=i;
end

%mengurutkan kromosom orang tua berdasarkan nilai fitness dari yang
besar sampai yang paling kecil

sementara1=sortrows([vektor_fitness vektor_parent]);

j=1;
for k=0:sizeparent-1
   sementara2(j,:)=sementara1(sizeparent-k,:);
   j=j+1;
end

r=sizeparent-sizechild;

orangtua=zeros(size(parent,1),r);

for l=1:r
   orangtua(:,l)=parent(:,sementara2(l,2));
end

populasi=[child orangtua];

uk_populasi=size(populasi,2);
```

```
%menghitung nilai fitness populasi baru
for m=1:uk_populasi

populasi_fitness(m,:)=FITNESS(populasi(:,m),spektrum,panjang_fragmen
                              ,panjang_sequence);
   vektor_populasi(m,:)=m;
end

sementara=sortrows([populasi_fitness vektor_populasi]);

%nilai fitness terbaik pada tiap generasi
fitness_best=sementara(uk_populasi,1);
```

- **SEQUENCE**

```
function[seq_ind]=SEQUENCE(individu,spektrum,panjang_fragmen)

r=size(individu,1);
k=1;
%membaca nukleotida pada fragmen urutan pertama
for i=1:panjang_fragmen
     seq_ind(k,:)=spektrum(i,individu(1,:));
     k=k+1;
end
%membaca nukleotida pada fragmen urutan 2 sampai terakhir
j=2;
while j<=r
   q=OVERLAPP(individu(j-
1,:),individu(j,:),spektrum,panjang_fragmen);
   for n=1+q:panjang_fragmen
       seq_ind(k,:)=spektrum(n,individu(j,:));
       k=k+1;
   end
   j=j+1;
end
```

- **CETAK_SEQUENCE**

```
function[]=CETAK_SEQUENCE(individu,spektrum,max_oligo)

seq_ind=SEQUENCE(individu,spektrum,max_oligo);

uk_seq_ind=size(seq_ind,1);

for i=1:uk_seq_ind
    fprintf('%s',seq_ind(i,:));
```

```
end
```

- **SOLUSI**

```
function[]=SOLUSI(populasi,spektrum,panjang_fragmen,...
                  panjang_sequence)

uk_populasi=size(populasi,2);

%menghitung nilai fitness populasi baru
for i=1:uk_populasi
      vektor_fitness(i,:)=FITNESS1(populasi(:,i),spektrum,...
                                   panjang_fragmen,panjang_sequence);
      vektor_populasi(i,:)=i;
end

sementara=sortrows([vektor_fitness vektor_populasi]);

best_ind=populasi(:,sementara(uk_populasi,2));
solusi_ind=best_ind'
CETAK_SEQUENCE(best_ind,spektrum,panjang_fragmen)
fprintf('\n');
fprintf('best subsequence :');
best_sub_sequence=BEST(best_ind,spektrum,panjang_fragmen,panjang_seq
uence);
CETAK_SEQUENCE(best_sub_sequence,spektrum,panjang_fragmen)
fprintf('\n');
```

- **BEST**

```
function[best_sub_sequence]=BEST(individu,spektrum,...
                                 panjang_fragmen,panjang_sequence)

nilai_fitness=0;
a=size(individu,1);
uk_vek=1;

oligo=1;
panjang=panjang_fragmen;

while oligo<=a
   for i=oligo+1:a
      b=OVERLAPP(individu(i-1,:),individu(i,:),...
                 spektrum,panjang_fragmen);
      panjang=panjang+panjang_fragmen-b;
      if panjang>panjang_sequence
         vek_fitness(uk_vek,:)=i-oligo;
         vek_oligo(uk_vek,:)=uk_vek;
         uk_vek=uk_vek+1;
```

```
         break;
      end
   end
   if panjang<=panjang_sequence
      vek_fitness(uk_vek,:)=i-oligo+1;
      vek_oligo(uk_vek,:)=uk_vek;
      uk_vek=uk_vek+1;
      break;
   end
   oligo=oligo+1;
end
sementara=sortrows([vek_fitness vek_oligo]);
uk_vek1=size(sementara,1);

j=1;
for
i=sementara(uk_vek1,2):sementara(uk_vek1,1)+sementara(uk_vek1,2)-1
    best_sub_sequence(j,:)=individu(i,:);
    j=j+1;
end
```

- **BENTUK_SPEKTRUM**

```
function[spektrum]=BENTUK_SPEKTRUM(barisan,panjang_fragmen,...
                                   panjang_sequence)

kolom=1;
for m=1:panjang_sequence-panjang_fragmen+1
    baris=1;
    for n=m:m+panjang_fragmen-1
        if barisan(n,:)==1
            spektrum(baris,kolom)='A';
        end
        if barisan(n,:)==2
            spektrum(baris,kolom)='T';
        end
        if barisan(n,:)==3
            spektrum(baris,kolom)='C';
        end
        if barisan(n,:)==4
            spektrum(baris,kolom)='G';
        end
        baris=baris+1;
    end
    kolom=kolom+1;
end
```

- **IDEAL**

```
function[nilai]=IDEAL(spektrum)

uk_kolom=size(spektrum,2);
nilai=1;

for i=1:uk_kolom
    for j=i+1:uk_kolom
        if spektrum(:,i)==spektrum(:,j)
            nilai=0;
            break;
        end
    end
end
```

LAMPIRAN 2

Format *File Input*

25
20 6
T T A A C G
T A A C G C
A A C G C A
A C G C A T
C G C A T T
G C A T T A
C A T T A T
A T T A T T
T T A T T C
T A T T C C
A T T C C G
T T C C G A
T C C G A A
C C G A A T
C G A A T A
G A A T A G
A A T A G G
A T A G G T
T A G G T T
A G G T T C

Baris pertama menyatakan panjang barisan DNA yang akan dicari. Baris kedua secara berturut-turut menyatakan banyak fragmen dan panjang fragmen. Kemudian baris-baris selanjutnya menyatakan fragmen anggota spektrum.