PENERAPAN ALGORITMA GENETIKA PADA PEWARNAAN TITIK SUATU GRAF

SKRIPSI

Oleh :
I WAYAN SRI SMERTIKA ADHI
0510940024-94

PROGRAM STUDI MATEMATIKA
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG
2011

PENERAPAN ALGORITMA GENETIKA PADA PEWARNAAN TITIK SUATU GRAF

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana Sains dalam bidang Matematika

Oleh :
I WAYAN SRI SMERTIKA ADHI
0510940024-94

PROGRAM STUDI MATEMATIKA
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2011


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

PENERAPAN ALGORITMA GENETIKA PADA PEWARNAAN TITIK SUATU GRAF

Oleh:
I WAYAN SRI SMERTIKA ADHI
0510940024-94

Setelah dipertahankan di depan Majelis Penguji pada tanggal 4 Februari 2011 dan dinyatakan memenuhi syarat untuk memperoleh gelar Sarjana Sains dalam bidang Matematika

Pembimbing I	Pembimbing II
Drs. Marsudi, MS	**Dra.Endang Wahyu H.,MSi**
NIP. 196101171988021002	**NIP. 196611121991032001**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf A., MSc
NIP. 196709071992031001

UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**: I Wayan Sri Smertika Adhi**
NIM	**: 0510940024**
Jurusan	**: Matematika**
Penulis Skripsi berjudul	**: Penerapan Algoritma Genetika pada Pewarnaan Titik suatu Graf**

Dengan ini menyatakan bahwa :

1. **Skripsi ini adalah benar-benar karya saya sendiri dan bukan hasil plagiat dari karya orang lain. Karya-karya yang tercantum dalam Daftar Pustaka, semata-mata digunakan sebagai acuan.**
2. **Apabila di kemudian hari diketahui bahwa isi Skripsi saya merupakan hasil plagiat, maka saya bersedia menanggung akibat hukum dari keadaan tersebut.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 4 Februari 2011
Yang menyatakan,

(I Wayan Sri Smertika Adhi)
NIM 0510940024

PENERAPAN ALGORITMA GENETIKA
PADA PEWARNAAN TITIK SUATU GRAF

ABSTRAK

Skripsi ini membahas penerapan algoritma genetika pada pewarnaan titik suatu graf. Pewarnaan titik graf menggunakan algoritma genetika dilakukan dengan terlebih dahulu merepresentasikan kromosom sebagai sekumpulan gen dengan nilai acak yang dikodekan ke dalam bilangan asli (1,2,3,...,*n* atau 1,2,3,...,*k*), di mana *n* adalah jumlah titik dan *k* adalah jumlah warna yang dimasukkan. Pada akhir generasi maksimum ditentukan kromosom solusi dengan nilai *fitness* maksimum dan total *conflict gen* minimum.

Pengaruh parameter genetika terhadap solusi pewarnaan titik graf *G* (n = 50) yaitu peningkatan nilai generasi maksimum dan jumlah populasi (i) mengakibatkan meningkatnya waktu komputasi. Peningkatan nilai generasi maksimum juga mengakibatkan solusi pewarnaan titik konvergen, sedangkan peningkatan jumlah populasi (i) mengakibatkan bertambahnya variasi solusi. Peningkatan nilai peluang mutasi (P_m) dan peluang *crossover* (P_c) berpengaruh pada peminimuman total *conflict gen*, namun tidak mengakibatkan waktu komputasi meningkat.

Kata kunci : Algoritma Genetika, Pewarnaan Titik Graf, Parameter genetika.

AN IMPLEMENTATION OF GENETIC ALGORITHM AT THE VERTEX OF GRAPH COLORING

ABSTRACT

This final project discusses the implementation of genetic algorithm at the vertex of graph coloring. Vertex graph coloring using genetic algorithm performed by first representing the chromosome as a set of genes with random numbers are encoded into natural numbers (1,2,3,...,*n* or 1,2,3,...,*k*), where *n* is the number of vertex and *k* is the number of colors included. At the end of the maximum generation is determined solution of chromosome with maximum fitness point and minimum total gene conflict.

The effects of genetic parameters in the graph vertex coloring solution G ($n = 50$) which increased the maximum generation point and the number of population (*i*) result in increased computational time. Increasing the point of the maximum generation also resulted in vertex coloring solution is convergent, while an increasing number of population (*i*) result in increased variation of the solution. Increasing the value of chance mutation (P_m) and crossover probability (P_c) influence on minimizing conflict genes numbers, but did not result in increased computational time.

Keywords : Genetic Algorithm, Graph Vertex Coloring, Genetic Parameters.

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Ida Sang Hyang Widhi Wasa, karena kasih, berkat, anugerah, hikmat dan kebijaksanaan yang diberikanNya, Skripsi yang berjudul "**Penerapan Algoritma Genetika pada Pewarnaan Titik suatu Graf**" ini dapat diselesaikan. Skripsi ini disusun dan diajukan sebagai syarat untuk memperoleh gelar Sarjana Sains dalam bidang Matematika di Universitas Brawijaya Malang.

Dukungan, motivasi dan bimbingan dari berbagai pihak sangat membantu dalam penulisan skripsi ini. Untuk itu penulis menyampaikan terima kasih yang sebesar-besarnya kepada:

1. Drs. Marsudi, MS selaku dosen pembimbing I dan Dra. Endang Wahyu H., MSi selaku dosen pembimbing II atas bimbingan dan arahan sehingga penulis dapat menyelesaikan Skripsi ini dengan baik.
2. Dr.Wuryansari Muharini K., MSi, Drs. Imam Nurhadi P., MT, Syaiful Anam, SSi., MT selaku dosen penguji atas segala saran yang diberikan untuk perbaikan skripsi ini.
3. Orang tua dan Saudaraku. Terima kasih atas cinta, kasih sayang, doa, dukungan dan semangat yang tiada henti.
4. Gema, Iwan, Fiqih, Hadi, Yopi, Aji, Dede, teman seperjuangan MATH'05, teman futsal yang selalu memberi gangguan dan semangat.
5. Semua pihak yang telah membantu terselesaikannya skripsi ini yang tidak bisa penulis sebutkan satu-persatu.

Akhir kata penulis berharap skripsi ini dapat memberikan sumbangan pemikiran yang bermanfaat bagi semua pihak yang membutuhkan. Penulis menyadari akan keterbatasan dalam menyelesaikan skripsi ini sehingga banyak kekurangan dan belum dapat dikatakan sempurna. Dengan demikian kritik dan saran yang membangun diharapkan dari semua pihak demi kesempurnaan skripsi ini.

Malang, 4 Februari 2011

Penulis

DAFTAR ISI

Halaman

DAFTAR TABEL

Halaman


UNIVERSITAS BRAWIJAYA

DAFTAR GAMBAR

Halaman

DAFTAR LAMPIRAN

Halaman

BAB I
PENDAHULUAN

1.1 Latar Belakang

Algoritma genetika pertama kali diperkenalkan di Universitas Michigan, Amerika Serikat oleh John Holland (1975) melalui sebuah penelitiannya yang kemudian dipopulerkan oleh salah satu muridnya, David Goldberg (1989). Goldberg mendefinisikan algoritma genetika sebagai metode algoritma pencarian berdasarkan pada mekanisme seleksi alam dan genetik alam.

Berbagai macam permasalahan dapat diselesaikan menggunakan algoritma genetika, di antaranya: penentuan nilai optimal suatu fungsi, pencarian jarak minimum *Traveling Salesman Problem* (TSP), penentuan solusi Catur Jawa, pewarnaan graf, dan berbagai permasalahan optimasi lainnya.

Terdapat tiga macam pewarnaan graf yaitu pewarnaan titik, pewarnaan sisi, dan pewarnaan daerah. Masalah utama dalam pewarnaan titik pada graf adalah bagaimana mewarnai semua titik pada graf sehingga tidak ada titik *adjacent* memiliki warna yang sama. Hal ini kemudian dikaitkan dengan penggunaan warna seminimum mungkin. Jumlah warna minimum yang dapat digunakan untuk mewarnai semua titik pada graf disebut dengan bilangan *kromatik* (Marsudi,1998).

Sebelumnya telah sering diterapkan berbagai metode konvensional dalam berbagai literatur untuk menyelesaikan masalah pewarnaan graf. Di antaranya adalah algoritma Welch-Powell, *Recursive Largest First* (RLF), dan metode konvensional lainnya, namun semua metode di atas memiliki kelemahan jika digunakan untuk menyelesaikan masalah pewarnaan graf dengan jumlah titik yang banyak dan tingkat keterhubungan antar titik yang semakin rumit.

Salah satu solusi dalam menyelesaikan permasalahan di atas adalah menggunakan metode heuristik. Ciri metode heuristik adalah selalu menemukan solusi yang baik tetapi tidak harus yang terbaik, serta cepat dan mudah untuk diimplementasikan. Algoritma genetika merupakan suatu metode penyelesaian masalah yang didasarkan pada metode heuristik dan dapat diterapkan pada masalah pewarnaan titik suatu graf.

Pada skripsi ini membahas penerapan algoritma genetika pada pewarnaan titik suatu graf serta mengkaji pengaruh parameter genetika terhadap solusi pewarnaan titik yang diperoleh menggunakan program.

1.2 Rumusan Masalah

Berdasarkan latar belakang, maka rumusan masalah pada skripsi ini adalah sebagai berikut.

1. Bagaimana menentukan pewarnaan titik suatu graf menggunakan algoritma genetika?
2. Bagaimana pengaruh parameter algoritma genetika terhadap solusi pewarnaan titik suatu graf?

1.3 Batasan Masalah

Adapun batasan masalah dalam penulisan skripsi ini adalah graf yang digunakan merupakan graf terhubung sederhana dan tak berarah dengan jumlah titik minimum 3.

1.4 Tujuan

Tujuan yang ingin dicapai dalam penulisan skripsi ini adalah

1. untuk menentukan cara pewarnaan titik suatu graf menggunakan algoritma genetika.
2. untuk mengetahui pengaruh parameter algoritma genetika terhadap solusi pewarnaan titik suatu graf.

BAB II
TINJAUAN PUSTAKA

Pada bab ini disajikan konsep dasar graf, pewarnaan titik pada graf, dan algoritma genetika. Beberapa tinjauan pustaka tersebut disajikan secara berurutan dan digunakan sebagai dasar dalam penyelesaian masalah pewarnaan titik suatu graf menggunakan algoritma genetika, di mana graf sebagai objek permasalahan dengan pewarnaan titik sebagai masalah yang akan diselesaikan dan algoritma genetika merupakan metode yang digunakan.

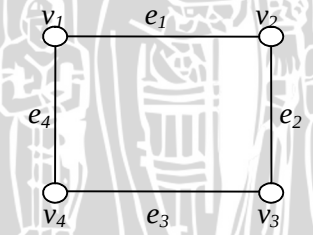
2.1 Konsep Dasar Graf

Definisi 2.1.1 (Harju, 2007)

Suatu graf G adalah pasangan himpunan $(V(G), E(G))$ dengan $V(G)$ adalah himpunan berhingga tak kosong dari titik-titik (*vertex*) dan $E(G)$ adalah himpunan berhingga (boleh kosong) dari sisi-sisi (*edges*).

Contoh 2.1 :

Gambar 2.1 menunjukkan graf G, dengan himpunan titik $V(G) = \{ v_1, v_2, v_3, v_4 \}$ dan himpunan sisi $E(G) = \{ e_1, e_2, e_3, e_4 \} = \{(v_1,v_2),(v_2,v_3),(v_3,v_4),(v_4,v_1)\}$.



Gambar 2.1 Graf G

Definisi 2.1.2 (Wilson dan Watkins, 1990)

Jika dua titik v_1 dan v_2 dihubungkan oleh suatu sisi e, maka v_1 dan v_2 disebut terhubung langsung (*adjacent*).

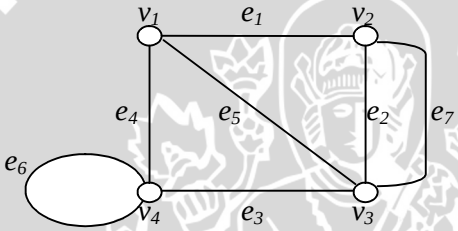
Contoh 2.2 :
Pada Gambar 2.1, graf *G* memuat pasangan titik *adjacent* yaitu (v_1, v_2), (v_2, v_3), (v_3, v_4), dan (v_4, v_1).

Definisi 2.1.3 (Harju, 2007)

Sisi yang menghubungkan suatu titik dengan dirinya sendiri disebut gelung (*loop*).

Contoh 2.3:

Gambar 2.2 menunjukkan graf *G* yang memuat gelung (*loop*) e_6.



Gambar 2.2 Graf *G* yang memuat gelung (*loop*) dan sisi ganda

Definisi 2.1.4 (Marsudi, 1998)

Misalkan *G* adalah graf tanpa *loop* dan *v* adalah titik di *G*. Derajat (*degree, valency*) dari *v* adalah banyaknya sisi yang terhubung dengan *v*, dan dilambangkan dengan *deg* (*v*).

Contoh 2.4 :
Pada Gambar 2.1, $deg(v_1) = deg(v_2) = deg(v_3) = deg(v_4) = 2$.

Definisi 2.1.5 (Harju, 2007)

Dua garis atau lebih yang menghubungkan pasangan titik yang sama disebut sisi ganda.

Contoh 2.5 :

Gambar 2.2 menunjukkan graf *G* yang memuat sisi ganda e_2 dan e_7 yang menghubungkan dua titik v_2 dan v_3.

Definisi 2.1.6 (Chartrand dan Zhang, 2005)

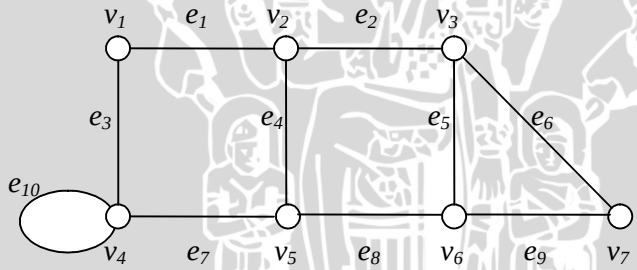
Jalan (*walk)* di *G* adalah barisan $v_0 e_1 v_1 e_2 v_2 ... v_{n-1} e_n v_n$ yang diberi lambang $v_0 - v_n$ dengan v_i adalah titik dan e_i adalah sisi yang menghubungkan titik v_{i-1} dan $v_i, i = 1,2,...,n.$

Definisi 2.1.7 (Marsudi, 1998)

Tapak *(trail)* adalah jalan yang semua sisinya berlainan, artinya yang diperhatikan adalah sisinya. Sementara itu lintasan (*path*) adalah jalan yang semua titiknya berbeda (tidak ada pengulangan titik yang sama).

Contoh 2.6 :

Gambar 2.3 menunjukkan graf *G* yang memuat jalan, lintasan, dan tapak. Jalan $v_1 - v_6$ adalah $v_1 e_1 v_2 e_2 v_3 e_6 v_7 e_6 v_3 e_5 v_6$ atau $v_1 e_1 v_2 e_2 v_3 e_5 v_6$, lintasan $v_1 - v_3$ adalah $v_1 e_1 v_2 e_2 v_3$ atau $v_1 e_3 v_4 e_7 v_5 e_8 v_6 e_5 v_3$, dan tapak $v_2 - v_1$ adalah $v_2 e_4 v_5 e_7 v_4 e_{10} v_4 e_3 v_1$.



Gambar 2.3 Graf *G* yang memuat jalan, lintasan, dan tapak

Definisi 2.1.8 (Wilson dan Watkins, 1990)

Graf sederhana (*simple graph*) adalah graf yang tidak memuat *loop* dan sisi ganda.

Contoh 2.7 :

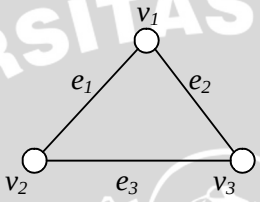
Pada Gambar 2.1, graf *G* merupakan graf sederhana karena tidak memuat *loop* dan sisi ganda. Pada Gambar 2.2, Graf *G* merupakan graf tidak sederhana karena memuat *loop* dan sisi ganda.

Definisi 2.1.9 (Marsudi, 1998)

Suatu graf di mana setiap pasang titik-titik yang berlainan *adjacent* disebut graf lengkap (*complete graph*). Graf lengkap dengan *n* titik dilambangkan dengan K_n.

Contoh 2.8 :

Pada Gambar 2.4, graf *G* merupakan graf lengkap dengan 3 titik dilambangkan dengan K_3.



Gambar 2.4 Graf lengkap *G* dengan jumlah titik 3

Definisi 2.1.10 Representasi Graf (Rosen, 2003)

Untuk setiap graf $G = (V(G), E(G))$ dengan $V(G) = \{v_1, v_2, ..., v_n\}$ dan $E(G) = \{e_1, e_2, ..., e_m\}$, matriks *adjacent* *A* dari *G* adalah matriks nol-satu berukuran *n x n* dengan 1 adalah entri ke-(*i*,*j*) menyatakan v_i dan v_j *adjacent,* dan 0 adalah entri ke-(*i*,*j*) menyatakan v_i dan v_j tidak *adjacent*.

Dengan kata lain, jika $A = [a_{ij}]$ matriks *adjacent* maka :

$$a_{ij} = \begin{cases} 1 & \text{jika } v_i \text{ } adjacent \text{ dengan } v_j \\ 0 & \text{jika } v_i \text{ tidak } adjacent \text{ dengan } v_j, \end{cases}$$

Contoh 2.9 :

Matriks *adjacent* pada graf *G*, Gambar 2.1 adalah :

$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

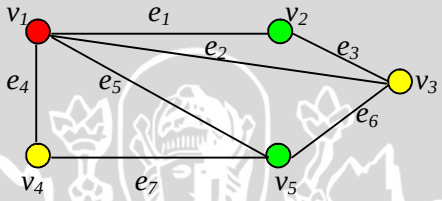
2.2 Pewarnaan Titik pada Graf

Definisi 2.2.1 (Marsudi, 1998)

Graf *G* (tanpa *loop*) dikatakan terwarnai-*k* (*k-colourable*) jika setiap titik dari *G* dapat diwarnai dengan satu dari *k* warna sedemikian sehingga setiap dua titik yang berdekatan mendapat warna yang berbeda.

Contoh 2.10 :

Gambar 2.5 menunjukkan pewarnaan titik pada graf *G* dengan lima titik dan terwarnai-3.



Gambar 2.5 Pewarnaan titik graf *G* dengan jumlah titik 5

Definisi 2.2.2 (Marsudi, 1998)

Graf *G* dikatakan ber*kromatik-k* (bilangan *kromatik G* adalah *k*) jika diperlukan paling sedikit *k* warna untuk mewarnai titik-titik dari *G* sedemikian sehingga tiap dua titik yang berdekatan mendapat warna yang berbeda. Dengan perkataan lain, graf *G* ber*kromatik-k* jika *G* terwarnai-*k* tetapi tidak terwarnai-(*k*-1). Dalam hal ini *k* disebut bilangan *kromatik*, ditulis $\lambda(G) = k$.

Definisi 2.2.3 (Marsudi, 1998)

Untuk setiap graf lengkap K_n berlaku $(\lambda(K_n)) = n$.

2.3 Algoritma Genetika (Fadlisyah dkk, 2009)

Algoritma Genetika adalah algoritma pencarian yang berdasarkan pada mekanisme alam yakni genetik dan seleksi alam. Berbeda dari teknik pencarian konvensional, algoritma genetika berangkat dari himpunan solusi yang dihasilkan secara acak. Himpunan ini disebut populasi. Sedangkan setiap individu dalam populasi disebut kromosom yang merupakan representasi dari solusi. Kromosom-kromosom berevolusi dalam suatu proses iterasi yang berkelanjutan yang disebut generasi. Pada setiap generasi, kromosom

dievaluasi berdasarkan suatu fungsi evaluasi. Setelah beberapa generasi maka algoritma genetika akan konvergen pada kromosom terbaik, yang diharapkan merupakan solusi optimal.

2.3.1 Karakteristik Algoritma Genetika (Fadlisyah dkk, 2009)

Goldberg (1989) mengemukakan bahwa algoritma genetika mempunyai karakteristik-karakteristik yang perlu diketahui sehingga dapat terbedakan dari prosedur pencarian atau optimasi yang lain, yaitu:

1. algoritma genetika menggunakan pengkodean dari himpunan solusi permasalahan berdasarkan parameter yang telah ditetapkan.
2. algoritma genetika merupakan pencarian sebuah solusi dari sejumlah individu-individu yang merupakan solusi permasalahan.
3. algoritma genetika memiliki informasi fungsi objektif (*fitness*) sebagai cara untuk mengevaluasi individu yang mempunyai solusi terbaik, bukan turunan dari suatu fungsi.
4. algoritma genetika menggunakan aturan-aturan transisi peluang, bukan aturan-aturan deterministik.

2.3.2 Variabel dan Parameter Algoritma Genetika (Fadlisyah dkk, 2009)

Variabel dan parameter yang digunakan pada algoritma genetika meliputi

1. fungsi *fitness* (fungsi tujuan) yang dimiliki oleh masing-masing individu untuk menentukan tingkat kesesuaian individu tersebut dengan kriteria yang ingin dicapai.
2. populasi jumlah individu yang dilibatkan pada setiap generasi.
3. probabilitas terjadinya persilangan (*crossover*) pada suatu generasi.
4. probabilitas terjadinya mutasi pada setiap individu.
5. jumlah generasi maksimum yang akan dibentuk.

2.3.3 Struktur Algoritma Genetika (Fadlisyah dkk, 2009)

Secara umum struktur suatu algoritma genetika dapat didefinisikan dengan langkah-langkah sebagai berikut

1. Membangkitkan populasi awal
   Populasi awal dibangkitkan secara acak sehingga diperoleh solusi awal. Populasi itu terdiri dari sejumlah kromosom yang merepresentasikan solusi yang diinginkan.
2. Membentuk generasi baru
   Untuk membentuk generasi baru, digunakan operator reproduksi/ seleksi, *crossover* dan mutasi. Proses ini dilakukan berulang-ulang sehingga didapatkan jumlah kromosom yang cukup untuk membentuk generasi baru yang merepresentasikan solusi baru. Generasi baru ini dikenal dengan istilah anak (*offspring*).
3. Evaluasi solusi
   Pada tiap generasi, kromosom akan melalui proses evaluasi dengan menggunakan alat ukur yang dinamakan *fitness*. Nilai *fitness* suatu kromosom menggambarkan kualitas kromosom dalam populasi tersebut. Proses ini akan mengevaluasi setiap populasi dengan menghitung nilai *fitness* setiap kromosom dan mengevaluasinya sampai terpenuhi kriteria berhenti. Bila kriteria berhenti belum terpenuhi maka akan dibentuk lagi generasi baru dengan mengulangi langkah 2. Beberapa kriteria berhenti sering digunakan antara lain: berhenti pada generasi tertentu, berhenti setelah dalam beberapa generasi berturut-turut didapatkan nilai *fitness* tertinggi dan tidak berubah, berhenti pada *n* generasi dan tidak didapatkan nilai fitness yang lebih tinggi.

Nilai *fitness* ditentukan oleh nilai penalti yang menunjukkan jumlah pelanggaran kendala pada suatu kromosom. Semakin tinggi nilai *fitness* akan semakin besar kemungkinan kromosom tersebut terpilih ke generasi berikutnya. Jadi nilai penalti berbanding terbalik dengan nilai *fitness*, semakin kecil nilai penalti (jumlah pelanggaran) semakin besar nilai *fitness*nya.

2.3.4 Operator Genetik (Fadlisyah dkk, 2009)

2.3.4.1 *Crossover*

Crossover merupakan operator genetik utama, yang beroperasi pada dua kromosom dalam suatu waktu dan menghasilkan *offspring* (anak) dengan mengkombinasikan kedua fitur-fitur kromosom. Cara yang paling sederhana untuk melakukan *crossover* adalah dengan memilih suatu *cut-point* dan menghasilkan *offspring* dengan pengkombinasian segmen dari *parent* pertama yang berada di

sebelah kiri bit-bit *cut-point*, dengan segmen *parent* kedua yang berada di sebelah kanan bit-bit *cut-point*.

Contoh 2.11 :
Tabel 2.1 menyajikan proses *crossover* 1-titik pada 2 kromosom induk (*parent*) terpilih (baris 2 dan 3) menggunakan pengkodean bilangan asli, sedangkan pada baris 4 dan 5 menyajikan kromosom anak (*offspring*) hasil proses *crossover*.

Tabel 2.1 Proses *crossover* 1-titik

Kromosom	Nilai gen
Parent 1	**1 2 3 4** 5 6 7 8 9
Parent 2	4 7 5 8 **6 9 3 1 2**
Offspring 1	**1 2 3 4 6 9 3 1 2**
Offspring 2	4 7 5 8 5 6 7 8 9

2.3.4.2 Mutasi

Mutasi merupakan sebuah operator *background* yang menghasilkan perubahan spontan secara acak pada berbagai kromosom. Cara yang paling sederhana dalam melakukan mutasi adalah dengan menukar satu atau lebih gen-gen pada satu kromosom. Dalam algoritma genetika, mutasi melakukan peran vital, yaitu menempatkan kembali gen-gen yang hilang dari populasi sepanjang proses seleksi agar mereka dapat dilibatkan kembali pada konteks yang berikutnya, dan menyediakan gen-gen yang tidak hadir pada populasi awal.

Contoh 2.12 :
Tabel 2.2 menyajikan proses mutasi pada 1 kromosom terpilih (baris 2) menggunakan pengkodean bilangan asli, sedangkan baris 3 menyajikan kromosom hasil proses mutasi.

Tabel 2.2 Proses mutasi

Keterangan	Nilai gen
Kromosom sebelum mutasi	1 2 **3** 4 6 5 8 **7** 9
Kromosom setelah mutasi	1 2 **7** 4 6 5 8 **3** 9

2.3.5 Operator Evolusi (Seleksi) (Fadlisyah dkk, 2009)

Proses seleksi menjamin bahwa individu atau kromosom dengan kualitas yang lebih baik akan lebih berpeluang untuk terpilih menjadi orang tua pada proses *crossover* daripada individu yang mempunyai kualitas lebih rendah.

2.3.5.1 Seleksi *Roulette Wheel*

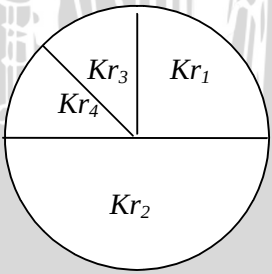
Sesuai dengan namanya, metode ini menirukan permainan *roulette-wheel* di mana masing-masing kromosom menempati potongan lingkaran pada roda *roulette* secara proporsional sesuai dengan nilai *fitness*nya. Kromosom yang memiliki nilai *fitness* lebih besar menempati potongan lingkaran yang lebih besar daripada kromosom bernilai *fitness* rendah.

Seleksi ini bertujuan untuk memberikan kesempatan reproduksi yang lebih besar bagi anggota populasi yang memiliki nilai *fitness* tinggi untuk bereproduksi.

Contoh 2.13 :

Tabel 2.3 Nilai *fitness* kromosom

Kromosom	**Nilai *fitness***
Kr_1	1
Kr_2	2
Kr_3	0.5
Kr_4	0.5
Jumlah	4



Gambar 2.6 Ilustrasi penggunaan metode *roulette-wheel selection*.

Kromosom dengan nilai *fitness* paling besar, menempati potongan sebesar setengah lingkaran. Dengan demikian, Kr_2 memiliki peluang sebesar 0,5 (2 dibagi 4) untuk terpilih sebagai orangtua.

2.3.5.2 Seleksi Rangking

Pada seleksi rangking, mula-mula dilakukan perangkingan kromosom dalam populasi, kemudian setiap kromosom mendapat nilai *fitness* berdasarkan dari rangking tersebut. Kromosom yang terbaik akan mendapatkan rangking 1, kedua terbaik mendapatkan rangking 2, dan seterusnya, kromosom terburuk akan mendapatkan rangking ke-*i* (jumlah kromosom dalam populasi).

Tabel 2.4 dan Tabel 2.5 menyajikan keadaan kromosom sebelum dirangking dan setelah mengalami perangkingan

Tabel 2.4 Keadaan sebelum dirangking

Populasi	*Fitness*
Kromosom 1	15
Kromosom 2	27
Kromosom 3	6
Kromosom 4	52
Kromosom 5	11

Tabel 2.5 Keadaan setelah dirangking

Populasi	*Fitness*	Rangking
Kromosom 4	52	1
Kromosom 2	27	2
Kromosom 1	15	3
Kromosom 5	11	4
Kromosom 3	6	5

2.3.6 Elitisme

Untuk menjaga agar kromosom dengan nilai *fitness* tertinggi tidak hilang selama evolusi, maka perlu dibuat satu atau beberapa *copy*-nya. Hal ini diperlukan karena tidak ada jaminan bahwa kromosom dengan nilai *fitness* tertinggi akan selalu terpilih, sebab seleksi induk dilakukan secara acak. Meskipun kromosom bernilai *fitness* tertinggi terpilih, masih ada kemungkinan kromosom tersebut akan rusak (nilai *fitness*nya menurun) karena proses genetika.

BAB III
HASIL DAN PEMBAHASAN

Pada bab ini akan ditunjukkan penerapan algoritma genetika untuk menentukan pewarnaan titik suatu graf serta mengkaji pengaruh parameter genetika terhadap solusi pewarnaan titik yang diperoleh menggunakan program.

3.1 Representasi Kromosom

Pada masalah pewarnaan titik graf, solusi yang ingin dicapai adalah semua titik terwarnai-*k* dengan pasangan titik yang *adjacent* menerima warna berbeda. Kromosom pada algoritma genetika untuk masalah pewarnaan graf menggunakan pengkodean bilangan asli (1,2,3,...,*k*) dengan masing-masing warna sebagai nilai gennya. Warna dikodekan ke dalam bilangan asli, dengan 1 merupakan warna-1 (c_1) sampai dengan *k* yang merupakan warna-*k* (c_k). Jumlah gen yang dibangkitkan bergantung pada jumlah titik graf yang dimasukkan. Untuk graf dengan *n* titik, maka kromosom yang dibangkitkan direpresentasikan sebagai Kr_i (v_1 v_2 v_3 ... v_n), di mana *i* adalah kromosom ke-*i* dan v_n adalah titik ke-*n* yang mewakili gen ke-*n* dengan nilai acak (1,2,3,...,*k*) warna.
Representasi kromosom dapat ditulis sebagai:

$$Kr_i(v_1 \quad v_2 \quad v_3 ... \quad v_n), all(v_n) \begin{cases} random(1,2,3,...,k) \\ random(1,2,3,...,n) \end{cases}$$

di mana $all(v_n)$ adalah *allele* (nilai gen) yang mewakili warna titik ke-*n* (v_n). Terdapat dua kriteria nilai $all(v_n)$, yaitu $random(1,2,3,...,k)$ jika jumlah warna diberikan dan $random(1,2,3,...,n)$ jika jumlah warna tidak diberikan.

3.1.1 *Conflict Gen* Kromosom

Pada masalah pewarnaan titik graf, untuk setiap graf *G* yang telah terwarnai-*k* dikatakan memiliki *conflict color* jika terdapat sepasang titik (*u*,*v*) atau lebih yang saling *adjacent* memiliki warna sama (*c(u) = c(v)*). Kromosom pada algoritma genetika yang menyatakan pewarnaan titik pada graf *G* dengan menggunakan pengkodean bilangan asli sebagai warna masing-masing titiknya, juga dapat dikatakan memiliki *conflict color* atau *conflict gen* jika

terdapat sepasang titik (*u*,*v*) atau lebih yang dinyatakan sebagai gen ke-*u* dan gen ke-*v* saling *adjacent* memiliki nilai gen sama. Dengan asumsi nilai gen telah diberikan terlebih dahulu pada gen ke-*u*, sedemikian hingga gen ke-*v* merupakan *conflict gen* pada kromosom *i*.

3.2 Fungsi *Fitness* atau Fungsi Objektif

Nilai *fitness* merupakan suatu ukuran baik tidaknya suatu solusi yang dinyatakan sebagai satu kromosom, atau dengan kata lain nilai *fitness* menyatakan nilai dari fungsi *fitness*. Algoritma genetika mempunyai tujuan untuk memaksimalkan nilai *fitness* atau mencari nilai *fitness* maksimum. Pada pembahasan ini, fungsi *fitness* dituliskan sebagai :

$$fitness = \frac{1}{f(p)}$$

f(p) merupakan fungsi penalti yang diperoleh dari :

$$f(p) = \sum_{(u,v)\in G} q(u,v) + d + C$$

di mana,

p adalah pewarnaan graf

q adalah nilai penalti untuk sepasang titik (*u*,*v*) yang saling *adjacent* dengan masing-masing warna titik (*c*(*u*),*c*(*v*)), nilai *q*(*u*,*v*) diperoleh dari :

$$q(u,v) = \begin{cases} 2, c(u) = c(v) \\ 0, c(u) \neq c(v) \end{cases}$$

d adalah nilai *general penalty* yang diperoleh dari:

$$d = \begin{cases} 1, \textit{ jika} \sum_{(u,v)\in G} q(u,v) > 0 \\ 0, \textit{ jika} \sum_{(u,v)\in G} q(u,v) = 0 \end{cases}$$

C adalah banyak warna yang digunakan

3.3 Elitisme

Pada masalah pewarnaan graf, proses elitisme dilakukan dengan meng*copy* dua kromosom teratas dengan nilai *fitness* maksimum dan total *conflict gen* minimum sebagai kromosom baru pada generasi berikutnya.

3.4 Seleksi

Seleksi dilakukan untuk memilih kromosom mana saja yang akan melalui proses *crossover* dan mutasi. Metode seleksi yang digunakan adalah metode seleksi roda *roulette*. Metode ini memberikan peluang terpilihnya kromosom dengan nilai *fitness* tertinggi semakin besar. Langkah metode seleksi roda *roulette* adalah sebagai berikut:

1. Menentukan nilai *fitness* dari masing-masing kromosom (*fitness*(Kr_i)).
2. Menentukan total *fitness* semua kromosom (F).

$$F = \sum_{i=1}^{n} fitness(Kr_i)$$

di mana, F : total *fitness*
fitness(Kr_i) : nilai *fitness* kromosom ke-i

3. Menentukan peluang seleksi (p_i) masing-masing kromosom.

$$p_i = \frac{fitness(Kr_i)}{F}, \quad i = 1,2,3,...,n$$

4. Menentukan peluang kumulatif (q_i) untuk masing-masing kromosom.

$$q_i = \sum_{i=1}^{n} p_i$$

5. Membangkitkan bilangan acak (r_{si} (bilangan acak seleksi ke-i)) antara 0 sampai 1 sebanyak i (sesuai jumlah kromosom yang melalui proses seleksi).
6. Menentukan kromosom yang terpilih dalam proses seleksi dengan ketentuan: jika ($q_{i-1} < r_{si} \leq q_i$) maka dipilih kromosom ke-i sebagai kromosom hasil seleksi.

3.5 *Crossover*

Metode *crossover* yang digunakan pada pembahasan ini adalah *conflict elimination crossover*. Metode ini bertujuan meminimalkan kesalahan pewarnaan (*conflict*) dengan cara saling menukarkan nilai gen induk (*parent*) yang mengalami *conflict* dan menghasilkan kromosom baru yang diharapkan memiliki kesalahan pewarnaan minimum. Langkah-langkah metode *conflict elimination crossover* adalah sebagai berikut:

1. Membangkitkan bilangan acak (r_{ci} (bilangan acak *crossover* ke-*i*)) antara 0 sampai 1 sebanyak *i* kromosom.
2. Menentukan kromosom yang akan mengalami proses *crossover* dengan memasukkan terlebih dahulu peluang *crossover* (P_c), jika $r_{ci} \leq P_c$ maka dipilih kromosom ke-*i* sebagai kromosom induk (*parent*) yang akan melalui proses *crossover*.
3. Jika banyak kromosom induk (*parent*) yang terpilih genap, maka proses *crossover* dilakukan pada setiap pasang kromosom induk (*parent*), sebaliknya jika jumlah kromosom induk (*parent*) yang terpilih ganjil, maka kromosom induk (*parent*) ke-*i* (induk terbawah yang masuk pada proses *crossover*) tidak akan mengalami proses *crossover* sehingga nilai gennya tidak mengalami perubahan.
4. Menukarkan nilai gen yang mengalami *conflict* antara induk (*parent*) pertama dengan induk (*parent*) kedua untuk menghasilkan kromosom baru (*Offspring*).

3.6 Mutasi

Metode mutasi yang digunakan pada pembahasan ini adalah *conflict free mutation*. Metode ini bertujuan meminimalkan kesalahan pewarnaan (*conflict*) dengan cara menggantikan nilai gen lama yang mengalami *conflict* dengan nilai gen acak baru untuk menghasilkan kromosom yang diharapkan memiliki kesalahan pewarnaan minimum. Langkah-langkah metode *conflict free mutation* adalah sebagai berikut:

1. Membangkitkan bilangan acak (r_{mi}(bilangan acak mutasi ke-*i*)) antara 0 sampai 1 sebanyak *i* kromosom.
2. Menentukan kromosom yang akan mengalami proses mutasi dengan memasukkan terlebih dahulu peluang mutasi (P_m),

jika $r_{mi} \leq P_m$ maka dipilih kromosom ke-*i* sebagai kromosom yang akan mengalami proses mutasi.

3. Mengganti nilai gen yang mengalami *conflict* dengan membangkitkan bilangan acak antara 1 sampai *n* titik atau 1 sampai *k* (jumlah warna yang dimasukkan, dengan $3 \leq k \leq 10$).

3.7 Langkah-langkah Penentuan Solusi Pewarnaan Titik Graf Menggunakan Algoritma Genetika

Langkah-langkah penentuan solusi pewarnaan titik graf menggunakan algoritma genetika adalah sebagai berikut:

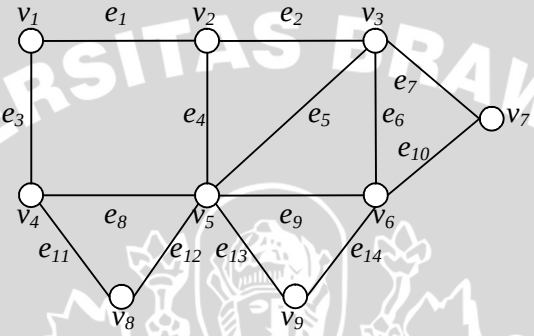
1. Diketahui graf terhubung sederhana dan tidak berarah dengan jumlah titik *n* dengan tiap titiknya diberi label (v_1,v_2,v_3,...,v_n). Kromosom yang dibangkitkan terdiri dari *n* barisan gen sesuai dengan jumlah titik graf yang diketahui.
2. Jika dimasukkan *k* warna, maka nilai gen dibangkitkan secara acak antara (1,2,3,...,*k*).
3. Menentukan matriks *adjacent* dari graf yang diketahui.
4. Membangkitkan *i* kromosom sebagai populasi awal.
5. Proses evaluasi nilai *fitness* kromosom.
   5.1 Menentukan kode warna masing-masing titik pada tiap kromosom ($c(v_n)$).
   5.2 Menentukan pasangan titik *adjacent* dan nilai fungsi penaltinya ($q(u,v)$).
   5.3 Menentukan *general penalty* (d).
   5.4 Menentukan jumlah warna yang digunakan (C).
   5.5 Menentukan nilai fungsi penalti ($f(p)$) dan nilai *fitness*.
   5.6 Menentukan pasangan titik *adjacent* dengan kode warna sama sebagai *conflict gen* dan jumlah gen yang mengakibatkan *conflict* (total *conflict gen*).
6. Proses elitisme.
   6.1 Mengurutkan kromosom berdasarkan nilai *fitness* dan total *conflict gen* (*fitness ranking*).
   6.2 Menyalin dua kromosom dengan rangking teratas sebagai kromosom baru pada generasi berikutnya.
7. Proses seleksi roda *roulette*.
   7.1 Menentukan nilai peluang seleksi (p_i) dan peluang kumulatif (q_i) untuk masing-masing kromosom yang tidak melalui proses elitisme (Kr_1').

7.2 Membangkitkan bilangan acak seleksi (r_{si}) pada selang [0,1] sesuai dengan banyak kromosom yang melalui proses seleksi.
7.3 Menentukan kromosom hasil seleksi.

8. Proses *crossover*.
   8.1 Menentukan kromosom induk yang akan di*crossover*.
   8.2 Proses *conflict elimination crossover*.
9. Menentukan *conflict gen* pada populasi baru hasil proses *crossover*.
   9.1 Menentukan kode warna masing-masing titik pada tiap kromosom ($c(v_n)$).
   9.2 Menentukan pasangan titik *adjacent* dan titik *conflict*nya.
   9.3 Menentukan pasangan titik *adjacent* yang memiliki kode warna sama sebagai *conflict gen*.
10. Proses mutasi.
    10.1 Menentukan kromosom yang akan dimutasi.
    10.2 Proses *conflict free mutation.*
11. Langkah 5 sampai 10 diulang secara terus-menerus hingga generasi mencapai generasi maksimum.
12. Proses evaluasi nilai *fitness* dan *fitness ranking* pada kromosom populasi akhir.
13. Menentukan kromosom solusi dengan nilai *fitness* tertinggi dan total *conflict gen* terendah sebagai solusi pewarnaan titik graf.

3.8 Contoh Kasus

Akan ditentukan pewarnaan titik pada graf G yang diberikan pada Gambar 3.1 menggunakan algoritma genetika jika diberikan tiga warna dan generasi maksimum sebesar 1 (diasumsikan jumlah populasi (i) sebesar 10, peluang *crossover* (P_c) sebesar 25%, peluang mutasi (P_m) sebesar 50%).



Gambar 3.1 Graf G dengan jumlah titik 9

Langkah-langkah pewarnaan titik graf G menggunakan algoritma genetika untuk satu generasi maksimum dengan jumlah titik 9 dan warna yang diberikan 3 adalah sebagai berikut.

Langkah 1: Berdasarkan Gambar 3.1, jumlah titik graf G adalah 9 di mana $V(G) = \{v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8,v_9\}$. Sehingga kromosom yang dibangkitkan terdiri dari 9 barisan gen sesuai dengan jumlah titik graf yang dimasukkan.

Langkah 2: Diberikan jumlah warna 3 yaitu c_1,c_2,c_3.

Misalkan: c_1 (*color*/warna 1) = merah
c_2 (*color*/warna 2) = biru
c_3 (*color*/warna 3) = hijau

Langkah 3: Menentukan matriks *adjacent* dari graf *G*.

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Langkah 4: Membangkitan *i* kromosom dengan nilai gen acak.

Dibangkitkan *i* = 10 kromosom dengan nilai gen acak (*all* ∈ (1,2,3)) menggunakan pengkodean bilangan asli:

1. Kr_1 (kromosom 1) = [3 1 2 3 1 2 3 2 1]
2. Kr_2 (kromosom 2) = [2 2 1 2 1 2 3 3 2]
3. Kr_3 (kromosom 3) = [3 3 1 2 1 3 3 1 2]
4. Kr_4 (kromosom 4) = [3 1 2 3 1 3 3 2 2]
5. Kr_5 (kromosom 5) = [1 1 2 1 1 3 2 3 3]
6. Kr_6 (kromosom 6) = [3 3 2 1 1 1 3 2 1]
7. Kr_7 (kromosom 7) = [1 2 3 2 2 3 1 3 1]
8. Kr_8 (kromosom 8) = [1 1 1 2 1 2 1 2 2]
9. Kr_9 (kromosom 9) = [1 2 3 3 2 3 3 3 1]
10. Kr_{10} (kromosom 10) = [1 3 2 3 1 1 1 3 3]

Langkah 5: Proses evaluasi nilai *fitness* kromosom.

Langkah-langkah proses evaluasi nilai *fitness* kromosom ke-1 (Kr_1 = [3 1 2 3 1 2 3 2 1]) adalah sebagai berikut:

Langkah 5.1 : Menentukan kode warna masing-masing titik pada tiap kromosom ($c(v_n)$).

Berdasarkan nilai gen acak dari kromosom ke-1 yang dibangkitkan, diperoleh warna masing-masing titik sebagai berikut:

1. $c(v_1) = c(v_4) = c(v_7) = c_3$
2. $c(v_2) = c(v_5) = c(v_9) = c_1$
3. $c(v_3) = c(v_6) = c(v_8) = c_2$

Langkah 5.2 : Menentukan pasangan titik *adjacent* dan nilai fungsi penaltinya ($q(u,v)$).

Tabel 3.1 menyajikan nilai fungsi penalti setiap pasangan titik *adjacent* (v_i,v_j) pada contoh kasus graf *G*. Fungsi penalti bernilai 2 jika pasangan titik (v_i,v_j) memiliki kode warna sama ($c(v_i) = c(v_j)$), sebaliknya fungsi penalti bernilai 0 jika pasangan titik (v_i,v_j) memiliki kode warna berbeda ($c(v_i) \neq c(v_j)$). Selanjutnya dapat ditentukan nilai $\sum_{(v_i,v_j)\in G} q(v_i,v_j)$ yang merupakan total nilai fungsi penalti pada Kr_i.

Tabel 3.1 Menentukan nilai fungsi penalti ($q(u,v)$) pada Kr_1

Pasangan titik *adjacent* (v_i,v_j)	**$c(v_i)$**	**$c(v_j)$**	**$q(v_i,v_j)$**
(v_1,v_2)	c_3	c_1	0
(v_1,v_4)	c_3	***c_3***	2
(v_2,v_3)	c_1	c_2	0
(v_2,v_5)	c_1	***c_1***	2
(v_3,v_5)	c_2	c_1	0
(v_3,v_6)	c_2	***c_2***	2
(v_3,v_7)	c_2	c_3	0
(v_4,v_5)	c_3	c_1	0
(v_4,v_8)	c_3	c_2	0
(v_5,v_6)	c_1	c_2	0
(v_5,v_8)	c_1	c_2	0
(v_5,v_9)	c_1	***c_1***	2
(v_6,v_7)	c_2	c_3	0
(v_6,v_9)	c_2	c_1	0
$\sum_{(v_i,v_j)\in G} q(v_i,v_j)$			8

Langkah 5.3 : Menentukan *general penalty* (*d*).

Berdasarkan Tabel 3.1, diperoleh nilai $\sum_{(v_i,v_j)\in G} q(v_i, v_j) > 0$, sehingga nilai *general penalty* (*d*) = 1.

Langkah 5.4 : Menentukan jumlah warna yang digunakan (*C*).

Jumlah warna yang dibangkitkan sebanyak tiga, sehingga *C* = 3.

Langkah 5.5 : Menentukan nilai fungsi penalti (*f*(*p*)) dan nilai *fitness*.

$$f(p) = \sum_{(v_i,v_j)\in G} q(v_i, v_j) + d + C$$

$$= 8 + 1 + 3$$

$$= 12$$

$$Fitness = \frac{1}{f(p)} = \frac{1}{12} = 0.08333333$$

Langkah 5.6 : Menentukan pasangan titik *adjacent* yang memiliki kode warna sama sebagai *conflict gen* dan jumlah gen yang mengalami *conflict* (total *conflict gen*).

Berdasarkan Tabel 3.1, terdapat empat pasang titik (v_i,v_j) yang mengalami *conflict*: (v_1,v_4), (v_2,v_5), (v_3,v_6), dan (v_5,v_9). Dari keempat pasang titik tersebut dapat ditentukan titik ke-*j* (v_j) sebagai titik *conflict*: v_4,v_5,v_6,v_9. Hal ini disebabkan kode warna *k* telah diberikan terlebih dahulu pada titik v_i (diasumsikan v_i memperoleh kode warna *k* terlebih dahulu sebelum v_j). Sehingga dapat ditentukan nilai gen pada Kr_1 yang mengalami *conflict* yaitu nilai gen ke-4, ke-5, ke-6, dan ke-9. Diperoleh:

Conflict gen Kr_1 = [3 1 2 **3 1 2** 3 2 **1**], total *conflict gen* = 4

Dari keseluruhan proses evaluasi kromosom diperoleh:

1. *fitness*(Kr_1 = [3 1 2 **3 1 2** 3 2 **1**]) = 0.083333333,
   total *conflict gen* = 4
2. *fitness* (Kr_2 = [2 **2** 1 **2 1** 2 3 3 **2**]) = 0.083333333,
   total *conflict gen* = 4
3. *fitness* (Kr_3 = [3 **3** 1 2 **1** 3 **3 1** 2]) = 0.083333333,
   total *conflict gen* = 4
4. *fitness* (Kr_4 = [3 1 2 **3 1** 3 **3** 2 2]) = 0.1,
   total *conflict gen* = 3

5. *fitness* (Kr_5 = [1 **1** 2 **1 1** 3 **2** 3 **3**]) = 0.0625, total *conflict gen* = 5
6. *fitness* (Kr_6 = [3 **3** 2 1 **1 1** 3 2 **1**]) = 0.0714286, total *conflict gen* = 4
7. *fitness* (Kr_7 = [1 2 3 2 **2 3** 1 3 1]) = 0.1, total *conflict gen* = 2
8. *fitness* (Kr_8 = [1 **1 1** 2 **1** 2 **1 2 2**]) = 0.0588235, total *conflict gen* = 6
9. *fitness* (Kr_9 = [1 2 3 3 **2 3 3 3** 1]) = 0.0714286, total *conflict gen* = 4
10. *fitness* (Kr_{10} = [1 3 2 3 1 **1 1 3** 3]) = 0.1, total *conflict gen* = 3

Langkah 6: Proses elitisme.

Langkah 6.1: *Fitness ranking*.

Tabel 3.2 menyajikan pengurutan kromosom berdasarkan nilai *fitness* dan total *conflict gen*. Kromosom diurutkan terlebih dahulu berdasarkan nilai *fitness*nya. Jika nilai *fitness* antara kromosom satu dengan kromosom lainnya sama maka langkah selanjutnya adalah membandingkan total *conflict gen*. Kromosom dengan nilai *fitness* tertinggi dan total *conflict gen* terendah akan menempati rangking teratas.

Tabel 3.2 *Fitness ranking* I

Kromosom	***Fitness*/ Total *conflict gen***	**Rangking**
Kr_7	0.1 / 2	1
Kr_4	0.1 / 3	2
Kr_{10} (1)	0.1 / 3	3
Kr_1 (2)	0.083333333 / 4	4
Kr_2 (3)	0.083333333 / 4	5
Kr_3 (4)	0.083333333 / 4	6
Kr_6 (5)	0.0714286 / 4	7
Kr_9 (6)	0.0714286 / 4	8
Kr_5 (7)	0.0625 / 5	9
Kr_8 (8)	0.0588235 / 6	10

Langkah 6.2: Penyalinan kromosom *elit*.

Pada proses elitisme dilakukan penyalinan dua kromosom dengan rangking teratas sebagai kromosom baru pada generasi berikutnya. Kromosom yang mengalami proses elitisme merupakan kromosom terbaik pada generasi pertama dan tidak akan melalui proses seleksi, proses *crossover* maupun proses mutasi.

Berdasarkan Tabel 3.2, diperoleh dua kromosom terbaik yang merupakan kromosom *elit*:

1. Kr_7 = [1 2 3 2 2 3 1 3 1]
2. Kr_4 = [3 1 2 3 1 3 3 2 2]

Selanjutnya diperoleh 8 kromosom yang tersisa, dilambangkan dengan Kr_i' (merupakan kromosom yang telah diurutkan dan tidak mengalami proses elitisme):

1. Kr_1' = (Kr_{10} = [1 3 2 3 1 **1 1 3** 3])
2. Kr_2' = (Kr_1 = [3 1 2 **3 1 2** 3 2 **1**])
3. Kr_3' = (Kr_2 = [2 **2** 1 **2 1** 2 3 3 **2**])
4. Kr_4' = (Kr_3 = [3 **3** 1 2 **1** 3 **3 1** 2])
5. Kr_5' = (Kr_6 = [3 **3** 2 1 **1 1** 3 2 **1**])
6. Kr_6' = (Kr_9 = [1 2 3 3 **2 3 3 3** 1])
7. Kr_7' = (Kr_5 = [1 **1** 2 **1 1** 3 **2** 3 **3**])
8. Kr_8' = (Kr_8 = [1 **1 1** 2 **1** 2 **1 2 2**])

Langkah 7: Proses seleksi roda *roulette*.

Langkah 7.1 : Menentukan nilai peluang seleksi (p_i) dan peluang kumulatif (q_i) untuk masing-masing kromosom (Kr_i').

Total *fitness i* populasi adalah:

$$F = \sum_{i=1}^{8} fitness(Kr_i') = 0.614181$$

Peluang seleksi (p_i) untuk masing-masing kromosom (Kr_i'), di mana i = 1,2,3,…,8, adalah sebagai berikut:

$$p_i = \frac{fitness(Kr_i')}{F}, i = 1,2,3,...,8$$

$$p_1 = \frac{fitness(Kr_1')}{F} = \frac{0.1}{0.614181} = 0.162818$$

$$p_2 = \frac{fitness(Kr_2')}{F} = \frac{0.083333}{0.614181} = 0.135682$$

$$p_3 = \frac{fitness(Kr_3')}{F} = \frac{0.083333}{0.614181} = 0.135682$$

$$p_4 = \frac{fitness(Kr_4')}{F} = \frac{0.083333}{0.614181} = 0.135682$$

$$p_5 = \frac{fitness(Kr_5')}{F} = \frac{0.0714286}{0.614181} = 0.116299$$

$$p_6 = \frac{fitness(Kr_6')}{F} = \frac{0.0714286}{0.614181} = 0.116299$$

$$p_7 = \frac{fitness(Kr_7')}{F} = \frac{0.0625}{0.614181} = 0.101762$$

$$p_8 = \frac{fitness(Kr_8')}{F} = \frac{0.0588235}{0.614181} = 0.095776$$

Peluang kumulatif (q_i) untuk masing-masing kromosom (Kr_i'), di mana i = 1,2,...,8, adalah sebagai berikut:

$$q_i = \sum_{j=1}^{i} p_j$$

$$q_1 = \sum_{j=1}^{1} p_j = p_1$$

$$= 0.162818$$

$$q_2 = \sum_{j=1}^{2} p_j = p_1 + p_2$$

$$= 0.162818 + 0.135682$$

$$= 0.2985$$

$$q_3 = \sum_{j=1}^{3} p_j = p_1 + p_2 + p_3$$

$$= 0.162818 + 0.135682 + 0.135682$$

$$= 0.434182$$

$$q_4 = \sum_{j=1}^{4} p_j = p_1 + p_2 + p_3 + p_4$$

$$= 0.162818 + 0.135682 + 0.135682 + 0.135682$$

$$= 0.569864$$

$$q_5 = \sum_{j=1}^{5} p_j = p_1 + p_2 + p_3 + p_4 + p_5$$

$$= 0.162818 + 0.135682 + 0.135682 + 0.135682 + 0.116299$$

$$= 0.686163$$

$$q_6 = \sum_{j=1}^{6} p_j = p_1 + p_2 + p_3 + p_4 + p_5 + p_6$$

$$= 0.162818 + 0.135682 + 0.135682 + 0.135682 + 0.116299 + 0.116299$$

$$= 0.802462$$

$$q_7 = \sum_{j=1}^{7} p_j = p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7$$

$$= 0.162818 + 0.135682 + 0.135682 + 0.135682 + 0.116299 + 0.116299 + 0.101762$$

$$= 0.904224$$

$$q_8 = \sum_{j=1}^{8} p_j = p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8$$

$$= 0.162818 + 0.135682 + 0.135682 + 0.135682 + 0.116299 + 0.116299 + 0.101762 + 0.095776$$

$$= 1$$

Langkah 7.2 : Membangkitkan bilangan acak seleksi (r_{si}) (sesuai dengan jumlah kromosom yang melalui proses seleksi) pada selang [0,1].

Pemutaran roda *roulette* (pembangkitan bilangan acak) dilakukan sebanyak 8 kali sesuai dengan jumlah kromosom yang melalui proses seleksi, dan pada masing-masing waktu dipilih satu

kromosom sebagai anggota populasi baru. Diasumsikan barisan delapan bilangan acak yang telah dibangkitkan pada interval [0,1] adalah sebagai berikut:

1. $r_{s1} = 0.881899$
2. $r_{s2} = 0.070375$
3. $r_{s3} = 0.777123$
4. $r_{s4} = 0.782984$
5. $r_{s5} = 0.143188$
6. $r_{s6} = 0.646518$
7. $r_{s7} = 0.833753$
8. $r_{s8} = 0.990415$

Langkah 7.3 : Menentukan kromosom hasil seleksi.

Jika ($q_{i-1} < r_{si} \leq q_i$) maka dipilih kromosom ke-*i* sebagai kromosom hasil seleksi.

Tabel 3.3 menyajikan penentuan kromosom hasil seleksi dengan masing-masing bilangan acak (r_{si}) pada interval [q_{i-1}, q_i].

Tabel 3.3 Pemilihan kromosom seleksi

Kromosom	r_{si}	Interval
Kr_1'	0.881899	($q_6 < r_{s1} \leq q_7$)
Kr_2'	0.070375	($r_{s2} \leq q_1$)
Kr_3'	0.777123	($q_5 < r_{s3} \leq q_6$)
Kr_4'	0.782984	($q_5 < r_{s4} \leq q_6$)
Kr_5'	0.143188	($r_{s5} \leq q_1$)
Kr_6'	0.646518	($q_4 < r_{s6} \leq q_5$)
Kr_7'	0.833753	($q_6 < r_{s7} \leq q_7$)
Kr_8'	0.990415	($q_7 < r_{s8} \leq q_8$)

Berdasarkan Tabel 3.3, diperoleh kromosom-kromosom yang terpilih pada proses seleksi, dilambangkan dengan Kr_i":

1. Kr_1" = (Kr_7' = [1 **1** 2 **1 1** 3 **2** 3 **3**])
2. Kr_2" = (Kr_1' = [1 3 2 3 1 **1 1 3** 3])
3. Kr_3" = (Kr_6' = [1 2 3 3 **2 3 3 3** 1])
4. Kr_4" = (Kr_6' = [1 2 3 3 **2 3 3 3** 1])
5. Kr_5" = (Kr_1' = [1 3 2 3 1 **1 1 3** 3])
6. Kr_6" = (Kr_5' = [3 **3** 2 1 **1 1** 3 2 **1**])
7. Kr_7" = (Kr_7' = [1 **1** 2 **1 1** 3 **2** 3 **3**])
8. Kr_8" = (Kr_8' = [1 **1 1** 2 **1** 2 **1 2 2**])

Langkah 8: Proses *crossover*.

Langkah 8.1: Menentukan kromosom induk yang akan di*crossover*

Dibangkitkan bilangan acak *crossover* (r_{ci}) dengan interval [0,1] untuk masing-masing kromosom (8 kromosom hasil proses seleksi).

Tabel 3.4 menyajikan kromosom hasil seleksi dengan masing-masing bilangan acak *crossover* (r_{ci}) yang telah dibangkitkan.

Tabel 3.4 Pembangkitan bilangan acak pada delapan kromosom induk

Kromosom	r_{ci}	Keterangan
Kr_1"	0.237357	$r_{c1} \leq 0.25$
Kr_2"	0.362229	$r_{c2} > 0.25$
Kr_3"	0.031341	$r_{c3} \leq 0.25$
Kr_4"	0.171077	$r_{c4} \leq 0.25$
Kr_5"	0.099763	$r_{c5} \leq 0.25$
Kr_6"	0.674317	$r_{c6} > 0.25$
Kr_7"	0.921531	$r_{c7} > 0.25$
Kr_8"	0.133719	$r_{c8} \leq 0.25$

Menentukan kromosom induk yang akan di*crossover* dengan ketentuan: Jika $r_{ci} \leq P_c$ maka dipilih kromosom ke-*i* sebagai kromosom induk yang akan mengalami proses *crossover*.

Berdasarkan Tabel 3.4, diperoleh kromosom induk terpilih untuk proses *crossover* yang disajikan pada Tabel 3.5:

Tabel 3.5 Kromosom induk yang terpilih untuk proses *crossover*

Kromosom	Nilai gen
Kr_3"	[1 2 3 3 **2 3 3 3** 1]
Kr_5"	[1 3 2 3 1 **1 1 3** 3]
Kr_8"	[1 **1 1** 2 1 2 **1 2 2**]
Kr_4"	[1 2 3 3 **2 3 3 3** 1]
Kr_1"	[1 **1** 2 **1 1** 3 **2** 3 **3**]

Langkah 8.2: Proses *conflict elimination crossover*

Tabel 3.6 Proses *conflict elimination crossover*

Parent	**Kromosom induk**	***Offspring***	**Kromosom anak**
Parent 1 (Kr_3")	[1 2 3 3 **2 3 3 3** 1]	*Offspring* 1	[1 2 3 3 *1 1 1 3* 1]
Parent 2 (Kr_5")	[1 3 2 3 1 **1 1 3** 3]	*Offspring* 2	[1 3 2 3 1 *3 3 3* 3]
Parent 3 (Kr_8")	[1 **1 1** 2 **1** 2 **1 2 2**]	*Offspring* 3	[1 *2 3* 2 *2* 2 *3 3 1*]
Parent 4 (Kr_4")	[1 2 3 3 **2 3 3 3** 1]	*Offspring* 4	[1 2 3 3 *1 2 1 2* 1]
Parent 5 (Kr_1")	[1 **1** 2 **1 1** 3 **2** 3 **3**]	*Offspring* 5	[1 *1* 2 *1 1* 3 *2* 3 *3*]

Berdasarkan Tabel 3.6, diperoleh empat kromosom anak (*offspring*) baru hasil persilangan gen induk yang mengalami *conflict*. Pada kromosom induk ke-5 (*parent* 5) tidak dilakukan proses *crossover* karena kromosom tersebut tidak memiliki pasangan untuk di-*crossover*, sehingga nilai gen di dalamnya tidak mengalami perubahan.

Langkah selanjutnya adalah penggabungan kromosom hasil *crossover* (sejumlah lima kromosom) dengan kromosom sebelumnya (tiga kromosom yang tidak melalui proses *crossover*). Sehingga diperoleh populasi baru dengan kromosom yang diberi tanda garis bawah sebagai kromosom baru hasil *crossover*.

1. Kr_1" = [1 1 2 1 1 3 2 3 3]
2. Kr_2" = [1 3 2 3 1 1 1 3 3]
3. Kr_3" = [1 2 3 3 1 1 1 3 1]
4. Kr_4 " = [1 2 3 3 1 2 1 2 1]
5. Kr_5 " = [1 3 2 3 1 3 3 3 3]
6. Kr_6 " = [3 3 2 1 1 1 3 2 1]
7. Kr_7 " = [1 1 2 1 1 3 2 3 3]
8. Kr_8 " = [1 2 3 2 2 2 3 3 1]

Langkah 9: Menentukan *conflict gen* pada populasi baru hasil proses *crossover*.

Langkah-langkah penentuan *conflict gen* kromosom ke-1 atau Kr_1" = [1 1 2 1 1 3 2 3 3] adalah sebagai berikut:

Langkah 9.1: Menentukan kode warna masing-masing titik pada tiap kromosom ($c(v_n)$).

Berdasarkan nilai gen pada kromosom ke-1 diperoleh warna masing-masing titik sebagai berikut:

1. $c(v_1) = c(v_2) = c(v_4) = c(v_5) = c_1$
2. $c(v_3) = c(v_7) = c_2$
3. $c(v_6) = c(v_8) = c(v_9) = c_3$

Langkah 9.2: Menentukan pasangan titik *adjacent* dan titik *conflict*nya.

Tabel 3.7 menyajikan pasangan titik *adjacent* pada graf *G* beserta kode warna masing-masing titiknya berdasarkan nilai gen pada kromosom Kr_1". Kode warna yang diberi tanda garis bawah merupakan kode warna yang mengakibatkan *conflict* pada titik v_j.

Tabel 3.7 Titik *conflict* pada Kr_1"

Pasangan titik *adjacent* (v_i,v_j)	**$c(v_i)$**	**$C(v_i)$**
(v_1,v_2)	c_1	$\underline{c_1}$
(v_1,v_4)	c_1	$\underline{c_1}$
(v_2,v_3)	c_1	c_2
(v_2,v_5)	c_1	$\underline{c_1}$
(v_3,v_5)	c_2	c_1
(v_3,v_6)	c_2	c_3
(v_3,v_7)	c_2	$\underline{c_2}$
(v_4,v_5)	c_1	$\underline{c_1}$
(v_4,v_8)	c_1	c_3
(v_5,v_6)	c_1	c_3
(v_5,v_8)	c_1	c_3
(v_5,v_9)	c_1	c_3
(v_6,v_7)	c_3	c_2
(v_6,v_9)	c_3	$\underline{c_3}$

Langkah 9.3: Menentukan pasangan titik *adjacent* yang memiliki kode warna sama kemudian diberikan tanda huruf tebal pada nilai gen yang mengakibatkan *conflict*.

Berdasarkan Tabel 3.7, terdapat enam pasang titik (v_i,v_j) yang mengalami *conflict*: (v_1,v_2), (v_1,v_4), (v_2,v_5), (v_3,v_7), (v_4,v_5), dan (v_6,v_9). Dari keenam pasang titik tersebut dapat ditentukan titik ke-*j* (v_j) sebagai titik *conflict*: v_2,v_4,v_5,v_7,v_9. Hal ini disebabkan kode warna *k* telah diberikan terlebih dahulu pada titik v_i (diasumsikan v_i memperoleh kode warna *k* terlebih dahulu sebelum v_j). Sehingga dapat ditentukan nilai gen pada Kr_1" yang mengalami *conflict* yaitu nilai gen ke-2, ke-4, ke-5, ke-7 dan ke-9. Diperoleh:

Conflict gen Kr_1" = [1 **1** 2 **1 1** 3 **2** 3 **3**]

Dari keseluruhan proses cek *conflict gen* kromosom diperoleh:

1. Kr_1" = [1 **1** 2 **1 1** 3 **2** 3 **3**]
2. Kr_2" = [1 3 2 3 1 **1 1 3** 3]
3. Kr_3" = [1 2 3 3 1 **1 1 3 1**]
4. Kr_4" = [1 2 3 3 1 2 1 2 **1**]
5. Kr_5" = [1 3 2 3 1 3 **3 3 3**]
6. Kr_6" = [3 **3** 2 1 **1 1** 3 2 **1**]
7. Kr_7" = [1 **1** 2 **1 1** 3 **2** 3 **3**]
8. Kr_8" = [1 2 3 2 **2 2 3** 3 1]

Langkah 10: Proses mutasi.

Langkah 10.1: Menentukan kromosom yang akan dimutasi.

Dibangkitkan bilangan acak mutasi (r_{mi}) dengan interval [0,1] untuk masing-masing kromosom (8 kromosom yang telah melalui proses *crossover*).

Tabel 3.8 menyajikan kromosom yang telah melalui proses *crossover* dengan masing-masing bilangan acak mutasi (r_{mi}) yang telah dibangkitkan.

Tabel 3.8 Pembangkitan bilangan acak pada delapan kromosom

Kromosom	r_{mi}	Keterangan
Kr_1"	0.537146	$r_{m1} > 0.5$
Kr_2"	0.235613	$r_{m2} \leq 0.5$
Kr_3"	0.083060	$r_{m3} \leq 0.5$
Kr_4"	0.098525	$r_{m4} \leq 0.5$
Kr_5"	0.417422	$r_{m5} \leq 0.5$
Kr_6"	0.455334	$r_{m6} \leq 0.5$
Kr_7"	0.447737	$r_{m7} \leq 0.5$
Kr_8"	0.517625	$r_{m8} > 0.5$

Menentukan kromosom yang akan dimutasi dengan ketentuan: Jika $r_{mi} \leq P_m$ maka dipilih kromosom ke-*i* sebagai kromosom yang akan mengalami proses mutasi.

Berdasarkan Tabel 3.8, diperoleh kromosom terpilih untuk proses mutasi yang selanjutnya disajikan pada Tabel 3.9.

Langkah 10.2: Proses *conflict free mutation*.

Tabel 3.9 menyajikan proses *conflict free mutation* pada kromosom terpilih. Nilai gen yang mengalami *conflict* (huruf tebal pada kolom 1) pada kromosom terpilih akan digantikan dengan nilai acak gen baru (huruf tebal dan bergaris bawah pada kolom 2), dengan nilai acak gen antara 1,2, dan 3 sesuai jumlah warna yang dimasukkan.

Tabel 3.9 Proses *conflict free mutation* pada kromosom terpilih

Kromosom awal	Kromosom setelah mutasi
Kr_2" = [1 3 2 3 1 **1 1 3** 3]	[1 3 2 3 1 **1 3 2** 3]
Kr_3" = [1 2 3 3 1 **1 1 3 1**]	[1 2 3 3 1 **2 1 2 1**]
Kr_4" = [1 2 3 3 1 2 1 2 **1**]	[1 2 3 3 1 2 1 2 **1**]
Kr_5" = [1 3 2 3 1 3 **3 3 3**]	[1 3 2 3 1 3 **2 2 2**]
Kr_6" = [3 **3** 2 1 **1 1** 3 2 **1**]	[3 **3** 2 1 **2 2** 3 2 **1**]
Kr_7" = [1 **1** 2 **1 1** 3 **2** 3 **3**]	[1 **3** 2 **2 1** 3 **1** 3 **3**]

Langkah selanjutnya adalah penggabungan kromosom hasil mutasi (sejumlah enam kromosom yang diberi tanda garis bawah) dengan kromosom sebelumnya (dua kromosom yang tidak melalui proses mutasi). Sehingga diperoleh populasi dengan kromosom baru:

1. Kr_1“ = [1 1 2 1 1 3 2 3 3]
2. Kr_2“ = [1 3 2 3 1 1 3 2 3]
3. Kr_3“ = [1 2 3 3 1 2 1 2 1]
4. Kr_4“ = [1 2 3 3 1 2 1 2 1]
5. Kr_5“ = [1 3 2 3 1 3 2 2 2]
6. Kr_6“ = [3 3 2 1 2 2 3 2 1]
7. Kr_7“ = [1 3 2 2 1 3 1 3 3]
8. Kr_8“ = [1 2 3 2 2 2 3 3 1]

Kemudian ditambahkan dengan dua salinan kromosom yang telah melalui proses elitisme (bergaris bawah). Sehingga diperoleh populasi baru yang selanjutnya akan digunakan pada generasi ke-2:

1. Kr_1 = [1 1 2 1 1 3 2 3 3]
2. Kr_2 = [1 3 2 3 1 1 3 2 3]
3. Kr_3 = [1 2 3 3 1 2 1 2 1]
4. Kr_4 = [1 2 3 3 1 2 1 2 1]
5. Kr_5 = [1 3 2 3 1 3 2 2 2]
6. Kr_6 = [3 3 2 1 2 2 3 2 1]
7. Kr_7 = [1 3 2 2 1 3 1 3 3]
8. Kr_8 = [1 2 3 2 2 2 3 3 1]
9. Kr_9 = [1 2 3 2 2 3 1 3 1]
10. Kr_{10} = [3 1 2 3 1 3 3 2 2]

Langkah 11: Proses generasi sudah mencapai generasi maksimum yaitu sebesar 1, sehingga langkah 5-10 tidak perlu diulang. Selanjutnya masuk pada langkah 12.

Langkah 12: Proses evaluasi nilai *fitness* dan *fitness ranking* pada kromosom populasi akhir.

Dari proses evaluasi nilai *fitness* seluruh kromosom pada populasi akhir diperoleh hasil:

1. *fitness* (Kr_1 = [1 **1** 2 **1 1** 3 **2** 3 **3**]) = 0.0625,
   total *conflict gen* = 5
2. *fitness* (Kr_2 = [1 3 2 3 1 **1** 3 2 3]) = 0.1666667,
   total *conflict gen* = 1
3. *fitness* (Kr_3 = [1 2 3 3 1 2 1 2 **1**]) = 0.1666667,
   total *conflict gen* = 1

4. *fitness* (Kr_4 = [1 2 3 3 1 2 1 2 **1**]) = 0.1666667,
   total *conflict gen* = 1
5. *fitness* (Kr_5 = [1 3 2 3 1 3 **2** 2 2]) = 0.1666667,
   total *conflict gen* = 1
6. *fitness* (Kr_6 = [3 **3** 2 1 **2 2** 3 **2** 1]) = 0.0714286,
   total *conflict gen* = 4
7. *fitness* (Kr_7 = [1 3 2 2 1 3 1 3 **3**]) = 0.1666667,
   total *conflict gen* = 1
8. *fitness* (Kr_8 = [1 2 3 2 **2 2 3** 3 1]) = 0.0833333,
   total *conflict gen* = 3
9. *fitness* (Kr_9 = [1 2 3 2 **2 3** 1 3 1]) = 0.1,
   total *conflict gen* = 2
10. *fitness* (Kr_{10} = [3 1 2 **3 1** 3 **3** 2 2]) = 0.1,
    total *conflict gen* = 3

Langkah selanjutnya adalah *fitness ranking* pada 10 kromosom di atas berdasarkan nilai *fitness* dan total *conflict gen* untuk memperoleh solusi terbaik dengan nilai *fitness* tertinggi dan total *conflict gen* terendah.

Tabel 3.10 menyajikan pengurutan kromosom berdasarkan nilai *fitness* dan total *conflict gen*. Kromosom diurutkan terlebih dahulu berdasarkan nilai *fitness*nya. Jika nilai *fitness* antara kromosom satu dengan kromosom lainnya sama maka langkah selanjutnya adalah membandingkan total *conflict gen*. Kromosom dengan nilai *fitness* tertinggi dan total *conflict gen* terendah akan menempati rangking teratas.

Tabel 3.10 *Fitness ranking* II

Kromosom	***Fitness***	**Total *conflict***	***Ranking***
Kr_2 = [1 3 2 3 1 **1** 3 2 3]	0.16666667	1	1
Kr_3 = [1 2 3 3 1 2 1 2 **1**]	0.16666667	1	2
Kr_4 = [1 2 3 3 1 2 1 2 **1**]	0.16666667	1	3
Kr_5 = [1 3 2 3 1 3 **2** 2 2]	0.16666667	1	4
Kr_7 = [1 3 2 2 1 3 1 3 **3**]	0.16666667	1	5
Kr_9 = [1 2 3 2 **2 3** 1 3 1]	0.1	2	6
Kr_{10} = [3 1 2 **3 1** 3 **3** 2 2]	0.1	3	7
Kr_8 = [1 2 3 2 **2 2 3** 3 1]	0.08333333	3	8
Kr_6 = [3 **3** 2 1 **2 2** 3 **2** 1]	0.0714286	4	9
Kr_1 = [1 **1** 2 **1 1** 3 **2** 3 **3**]	0.0625	5	10

Langkah 13: Menentukan kromosom solusi sebagai solusi pewarnaan titik graf.

Berdasarkan Tabel 3.10, diperoleh lima kromosom dengan nilai *fitness* tertinggi dan total *conflict gen* terendah yang merupakan solusi pewarnaan titik pada contoh kasus graf *G*. Karena nilai gen pada kromosom-3 (Kr_3) sama dengan nilai gen pada kromosom-4 (Kr_4) maka dapat disimpulkan hanya empat solusi yang diperoleh.

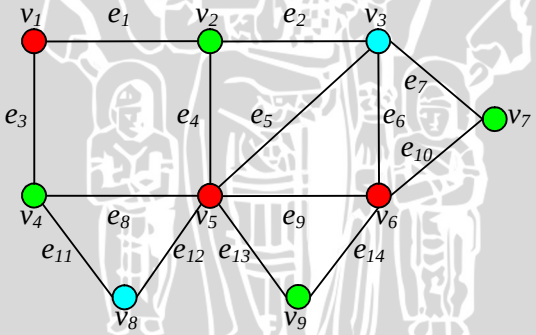
Solusi pewarnaan titik contoh kasus graf *G* menggunakan algoritma genetika untuk satu generasi maksimum dan jumlah warna yang diberikan 3, adalah sebagai berikut:

1. Solusi 1 (Kr_2 = [1 3 2 3 1 **1** 3 2 3]**)**

Berdasarkan nilai gen pada solusi 1 (Kr_2), diperoleh kode warna masing-masing titik graf *G*, yaitu:

$c(v_1) = c(v_5) = c(v_6) = c_1$
$c(v_2) = c(v_4) = c(v_7) = c(v_9) = c_3$
$c(v_3) = c(v_8) = c_2$

Untuk c_1 = merah, c_2 = biru, c_3 = hijau, maka dapat ditentukan solusi pewarnaan titik pertama seperti disajikan pada Gambar 3.2.



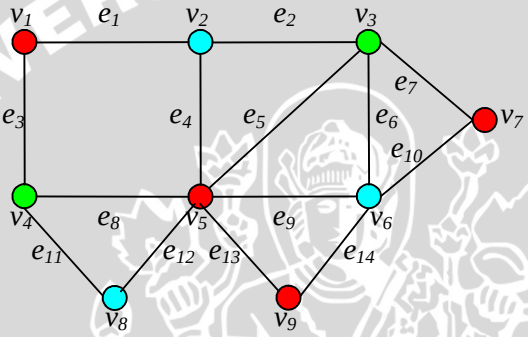
Gambar 3.2 Solusi 1 pewarnaan titik graf *G*

2. Solusi 2 (Kr_3 = [1 2 3 3 1 2 1 2 **1**]**).**

Berdasarkan nilai gen pada solusi 2 (Kr_3), diperoleh kode warna masing-masing titik graf G, yaitu:

$c(v_1) = c(v_5) = c(v_7) = c(v_9) = c_1$
$c(v_2) = c(v_6) = c(v_8) = c_2$
$c(v_3) = c(v_4) = c_3$

Untuk c_1 = merah, c_2 = biru, c_3 = hijau, maka dapat ditentukan solusi pewarnaan titik kedua seperti disajikan pada Gambar 3.3.



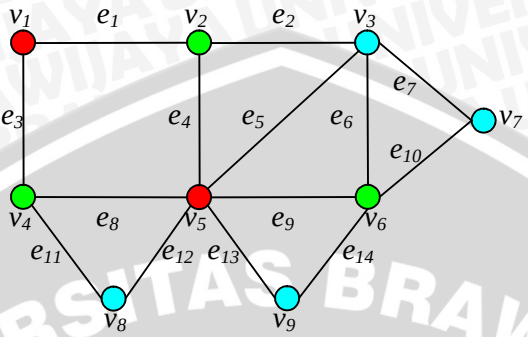
Gambar 3.3 Solusi 2 pewarnaan titik graf G

3. Solusi 3 (Kr_5 = [1 3 2 3 1 3 **2** 2 2]**).**

Berdasarkan nilai gen pada solusi 3 (Kr_5), diperoleh kode warna masing-masing titik graf G, yaitu:

$c(v_1) = c(v_5) = c_1$
$c(v_2) = c(v_4) = c(v_6) = c_3$
$c(v_3) = c(v_7) = c(v_8) = c(v_9) = c_2$

Untuk c_1 = merah, c_2 = biru, c_3 = hijau, maka dapat ditentukan solusi pewarnaan titik ketiga seperti disajikan pada Gambar 3.4.



Gambar 3.4 Solusi 3 pewarnaan titik graf *G*

4. Solusi 4 (Kr_7 = [1 3 2 2 1 3 1 3 **3**]**).**

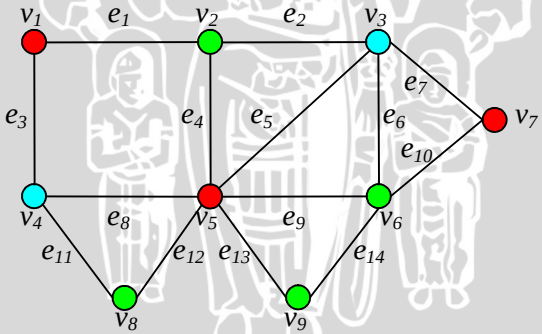
Berdasarkan nilai gen pada solusi 4 (Kr_7), diperoleh kode warna masing-masing titik graf *G*, yaitu:

$c(v_1) = c(v_5) = c(v_7) = c_1$

$c(v_2) = c(v_6) = c(v_8) = c(v_9) = c_3$

$c(v_3) = c(v_4) = c_2$

Untuk c_1 = merah, c_2 = biru, c_3 = hijau, maka dapat ditentukan solusi pewarnaan titik keempat seperti disajikan pada Gambar 3.5.

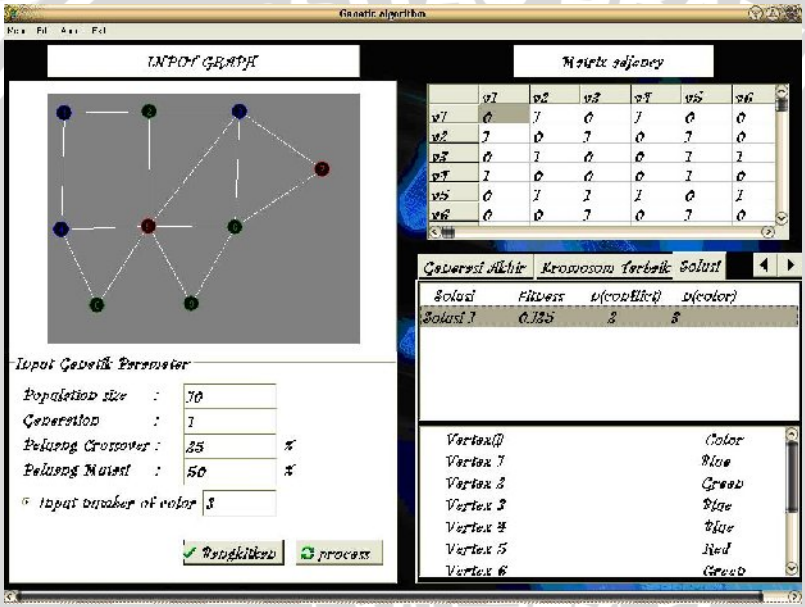


Gambar 3.5 Solusi 4 pewarnaan titik graf *G*

Berdasarkan keempat solusi pewarnaan titik graf *G* di atas masih ditemukan adanya *conflict color* pada masing-masing solusinya. Hal ini menunjukkan bahwa solusi yang dihasilkan menggunakan algoritma genetika untuk satu kali proses generasi tidak selalu menghasilkan solusi pewarnaan titik terbaik. Selanjutnya

akan ditunjukkan penyelesaian masalah pewarnaan titik graf *G* menggunakan program genetika dengan dua kriteria nilai parameter, yaitu sesuai dengan contoh kasus dan yang ditingkatkan nilai generasi maksimum serta jumlah populasinya.

Gambar 3.6 menyajikan solusi pewarnaan titik graf *G* menggunakan program genetika dengan nilai parameter yang sesuai pada contoh kasus, di mana i = 10, generasi maksimum = 1, P_c = 25%, P_m = 50%, C = 3.

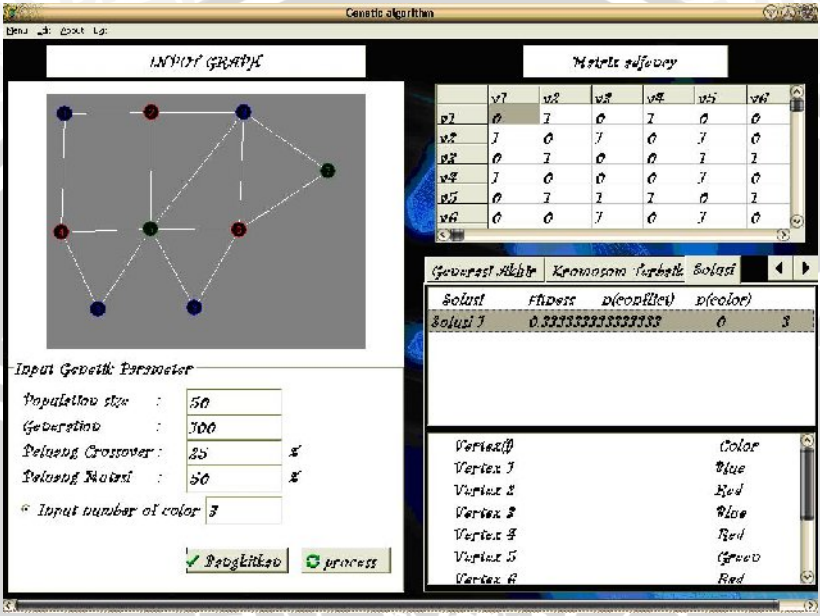


Gambar 3.6 Solusi komputasi 1 pewarnaan titik graf *G*

Berdasarkan Gambar 3.6, hanya diperoleh satu solusi pewarnaan titik pada graf *G*. Berbeda dengan solusi yang diperoleh pada perhitungan sebelumnya, hal ini disebabkan populasi awal yang dibangkitkan serta proses genetika yang dilakukan secara acak berbeda. Solusi komputasi pada Gambar 3.6 menunjukkan masih terdapat *conflict gen* sebesar 2 pada kromosom solusinya. Hal ini menunjukkan solusi pewarnaan titik di atas bukan merupakan solusi terbaik, karena masih ditemukannya *conflict color*.

Gambar 3.7 menyajikan solusi pewarnaan titik graf *G* menggunakan program komputasi genetika dengan parameter yang

ditingkatkan nilai generasi maksimum dan jumlah populasinya, yaitu: i = 50, P_c = 25%, P_m = 50%, C = 3, generasi maksimum = 100.



Gambar 3.7 Solusi komputasi 2 pewarnaan titik graf *G*

Berdasarkan Gambar 3.7, diperoleh satu solusi pewarnaan titik dengan total *conflict gen* pada kromosom solusinya sebesar 0. Dapat ditarik kesimpulan bahwa solusi pewarnaan titik di atas merupakan solusi pewarnaan terbaik dengan jumlah warna minimum dan total *conflict gen* sebesar 0. Solusi pewarnaan titik terbaik pada graf *G* diperoleh dengan meningkatkan nilai generasi maksimum dan jumlah populasi (*i*).

3.9 Implementasi Program Algoritma Genetika

Secara umum, sistem penyelesaian algoritma genetika untuk masalah pewarnaan titik graf dibuat menggunakan sistem komputer dengan spesifikasi sebagai berikut:

1. *Processor*: Intel(R) Core(TM) i3 M380 @2.53GHz, L2 *chace*: 32 KB,
2. *Memory*: 1912 MB,

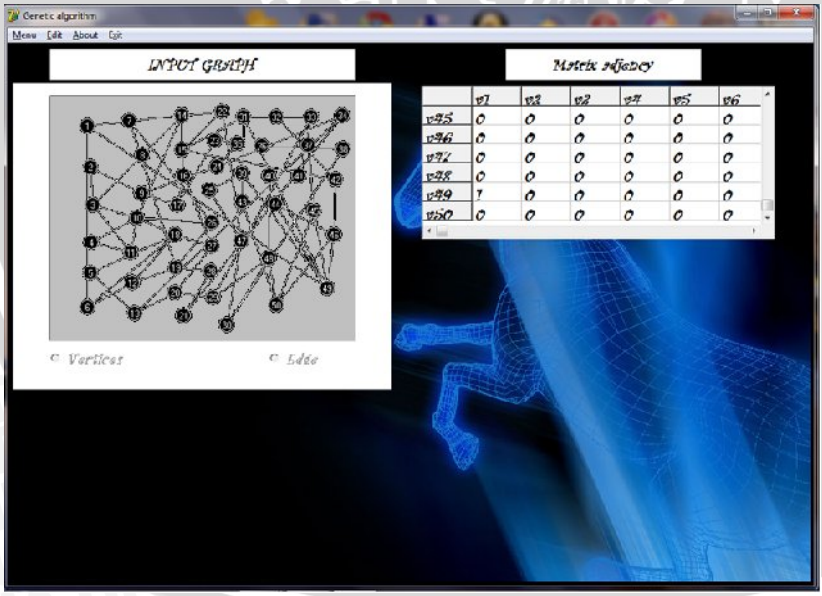
3. *Hard disk* berukuran 320 GB.

Sistem operasi yang digunakan adalah sistem operasi Microsoft Windows 7 *Home Premium*. Algoritma genetika diimplementasikan menggunakan perangkat lunak Borland Delphi 7.

3.10 Pengaruh Parameter Genetika Terhadap Solusi Pewarnaan Titik suatu Graf Berdasarkan Hasil dan Analisis Keluaran Program

Pada Subbab ini, program genetika menyelesaikan masalah pewarnaan titik graf sederhana dengan jumlah titik sebesar 50 dalam beberapa kali proses percobaan. Hal ini dilakukan sebab proses genetika berangkat dari himpunan solusi yang dihasilkan secara acak. Percobaan yang dilakukan adalah sebanyak lima perulangan untuk setiap kali program dijalankan. Dari hasil percobaan yang diperoleh, ditentukan perbandingan jumlah warna rata-rata, total *conflict gen* rata-rata, jumlah solusi rata-rata, dan waktu rata-rata dari nilai parameter yang berbeda.

Gambar 3.8 menyajikan graf sederhana dengan jumlah titik sebesar 50 serta matriks *adjacent*nya.



Gambar 3.8 graf G (n = 50)

Tabel 3.11 menyajikan perbandingan solusi pewarnaan titik graf G (n = 50) dengan perbedaan nilai generasi maksimum. Berdasarkan Tabel 3.11, peningkatan nilai generasi maksimum menyebabkan meningkatnya waktu komputasi. Hal ini disebabkan semakin besar generasi yang diproses oleh program genetika. Sedangkan untuk jumlah warna yang dihasilkan belum menunjukkan warna minimum. Pada nilai generasi maksimum sebesar 10 sampai 50 generasi, diperoleh jumlah warna sebesar 27, namun program genetika telah berhasil meminimalkan total *conflict* pada solusi pewarnaannya. Pada generasi maksimum sebesar 10, diperoleh dua solusi pewarnaan titik dengan total *conflict* sebesar 0.

Tabel 3.11 Perbandingan solusi pewarnaan titik graf G (n = 50) dengan nilai generasi maksimum berbeda

Parameter	***i* = 25, P_c =25%, P_m =50%**		
Generasi maksimum	**Jumlah warna (*C*) (rata-rata)**	**Total *conflict gen* (rata-rata) / Jumlah solusi (rata-rata)**	**Waktu (detik) (rata-rata)**
10	27	0 / 2	0.035879629634
20	27	0 / 1	0.053240740749
30	27	0 / 1	0.054398148153
40	27	0 / 1	0.072916666671
50	27	0 / 1	0.090277777776

Tabel 3.12 menyajikan perbandingan solusi pewarnaan titik graf G (n = 50) dengan perbedaan jumlah populasi (i). Berdasarkan Tabel 3.12, peningkatan jumlah populasi menyebabkan meningkatnya waktu komputasi. Hal ini disebabkan semakin besar jumlah populasi yang diproses oleh program genetika. Sedangkan untuk jumlah warna dan jumlah solusi yang dihasilkan, masing-masing menunjukkan hasil yang bervariasi dan semakin meningkat. Peningkatan jumlah populasi tidak menjamin jumlah warna yang dihasilkan akan semakin minimum (dengan asumsi nilai generasi maksimum tetap), hal ini disebabkan himpunan populasi awal yang akan diproses dibangkitkan secara acak, namun peningkatan jumlah populasi berpengaruh pada jumlah solusi pewarnaan yang dihasilkan, hal ini disebabkan semakin meningkatnya himpunan calon solusi yang dibangkitkan.

Tabel 3.12 Perbandingan solusi pewarnaan titik graf G (n = 50) dengan jumlah populasi (i) berbeda

Parameter	**Generasi maksimum = 10, P_c = 25%, P_m = 50%**		
Jumlah populasi (i)	**Jumlah warna (C) (rata-rata)**	**Total *conflict gen* (rata-rata) / Jumlah solusi (rata-rata)**	**Waktu (detik) (rata-rata)**
15	28	0 / 2	0.035879629623
25	29	0 / 2	0.037037037037
35	28	0 / 2	0.054398148153
45	27	0 / 2	0.054398148153
55	27	0 / 3	0.072916666660

Tabel 3.13 menyajikan perbandingan solusi pewarnaan titik graf G (n = 50) dengan perbedaan nilai peluang mutasi (P_m). Berdasarkan Tabel 3.13, peningkatan nilai peluang mutasi tidak berpengaruh pada perubahan waktu komputasi. Pada peningkatan nilai peluang mutasi sebesar 20% hingga 100% diperoleh waktu komputasi tetap sebesar 0.03588 detik. Peluang mutasi berpengaruh pada total *conflict gen* yang dihasilkan. Pada peluang mutasi sebesar 20% telah diperoleh total *conflict* sebesar 0.

Tabel 3.13 Perbandingan solusi pewarnaan titik graf G (n = 50) dengan nilai peluang mutasi (P_m) berbeda

Parameter	**Generasi maksimum = 10, i = 25, P_c = 25%**		
Peluang mutasi (P_m)	**Jumlah warna (C) (rata-rata)**	**Total *conflict gen* (rata-rata) / Jumlah solusi (rata-rata)**	**Waktu (detik) (rata-rata)**
20%	27	0 / 2	0.035879629634
40%	27	0 / 2	0.035879629634
60%	27	0 / 2	0.035879629623
80%	27	0 / 2	0.035879629623
100%	27	0 / 2	0.035879629623

Tabel 3.14 menyajikan perbandingan solusi pewarnaan titik graf G (n = 50) dengan perbedaan nilai peluang *crossover* (P_c). Berdasarkan Tabel 3.14, peningkatan nilai peluang *crossover* tidak berpengaruh pada perubahan waktu komputasi. Pada peningkatan

nilai peluang *crossover* sebesar 20% hingga 100% diperoleh waktu komputasi tetap sebesar 0.03588 detik. Peluang *crossover* berpengaruh pada total *conflict gen* yang dihasilkan. Pada peluang *crossover* sebesar 20% telah diperoleh total *conflict* sebesar 0.

Tabel 3.14 Perbandingan solusi pewarnaan titik graf G (n = 50) dengan nilai peluang *crossover* (P_c) berbeda

Parameter	**Generasi maksimum = 10, i = 25, P_m = 50%**		
Peluang *crossover* (P_c)	**Jumlah warna (C) (rata-rata)**	**Total *conflict gen* (rata-rata) / Jumlah solusi (rata-rata)**	**Waktu (detik) (rata-rata)**
20%	27	0 / 2	0.035879629634
40%	27	0 / 2	0.035879629623
60%	27	0 / 2	0.035879629623
80%	27	0 / 2	0.035879629634
100%	27	0 / 2	0.035879629623

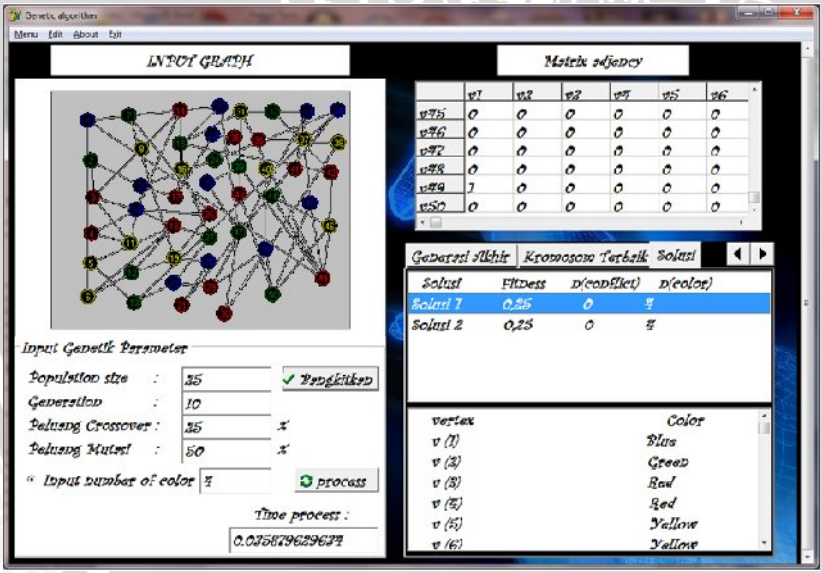
Berdasarkan perbandingan solusi pewarnaan titik graf G (n = 50) dari empat parameter yang berbeda, diperoleh solusi dengan total *conflict gen* sebesar 0. Hal ini menunjukkan bahwa algoritma genetika telah berhasil meminimalkan *conflict color* pada solusi pewarnaan titik yang diperoleh, namun berdasarkan jumlah warna solusi yang diperoleh, menunjukkan jumlah warna cukup besar yaitu sebesar 27 warna. Berdasarkan hasil tersebut dapat disimpulkan bahwa algoritma genetika tidak selalu menghasilkan solusi pewarnaan titik dengan jumlah warna minimum. Untuk mengatasi kelemahan tersebut, maka pada program genetika ditambahkan masukan berupa jumlah warna yang diinginkan. Selanjutnya pada Tabel 3.15 disajikan perbandingan solusi pewarnaan titik graf G (n = 50) dengan jumlah warna (C) berbeda. Solusi yang dihasilkan dibandingkan berdasarkan pada nilai total *conflict gen* dengan masing-masing jumlah warna yang diberikan.

Berdasarkan Tabel 3.15, diperoleh total *conflict gen* sebesar 0 pada jumlah warna sebesar 4. Peningkatan jumlah warna di atas empat tidak berpengaruh pada total *conflict gen* yang dihasilkan, di mana pada peningkatan tersebut diperoleh total *conflict gen* tetap sebesar 0. Dapat disimpulkan bahwa pewarnaan titik pada graf G (n = 50) yang diberikan memiliki jumlah warna minimum sebesar 4.

Tabel 3.15 Perbandingan solusi pewarnaan titik graf G (n = 50) dengan jumlah warna (C) berbeda

Parameter	**Generasi maksimum = 10, i = 25, P_c = 25%, P_m = 50%**	
Jumlah warna (C)	**Total *conflict gen* (rata-rata) / Jumlah solusi (rata-rata)**	**Waktu (detik) (rata-rata)**
3	4 / 1	0.035879629634
4	0 / 2	0.035879629634
5	0 / 3	0.035879629634
6	0 / 5	0.035879629634
7	0 / 5	0.035879629623

Gambar 3.9 menyajikan solusi pewarnaan titik minimum pada graf G (n = 50).



Gambar 3.9 Solusi pewarnaan titik graf G (n = 50, C = 4)

BAB IV
KESIMPULAN DAN SARAN

4.1 Kesimpulan

Berdasarkan hasil dan pembahasan yang diperoleh, dapat ditarik kesimpulan bahwa pewarnaan titik suatu graf menggunakan algoritma genetika dilakukan dengan terlebih dahulu merepresentasikan kromosom sebagai sekumpulan gen dengan nilai acak yang dikodekan ke dalam bilangan asli (1,2,3,...,n atau 1,2,3,...,k), di mana n adalah jumlah titik dan k adalah jumlah warna yang dimasukkan. Sebanyak i kromosom acak dibangkitkan, kemudian dilakukan proses genetika hingga dicapai generasi maksimum. Pada akhir generasi ditentukan kromosom solusi dengan nilai *fitness* maksimum dan total *conflict gen* minimum. Selanjutnya, pengaruh parameter genetika terhadap solusi pewarnaan titik graf G (n = 50) yang diberikan yaitu peningkatan nilai generasi maksimum dan jumlah populasi (i) masing-masing mengakibatkan meningkatnya waktu komputasi. Peningkatan nilai generasi maksimum juga mengakibatkan solusi pewarnaan titik konvergen, sedangkan peningkatan jumlah populasi (i) mengakibatkan bertambahnya variasi solusi. Peningkatan nilai peluang mutasi (P_m) dan peluang *crossover* (P_c) berpengaruh pada peminimuman total *conflict gen*, tetapi tidak mengakibatkan meningkatnya waktu komputasi.

4.2 Saran

Berdasarkan hasil kesimpulan yang diperoleh, terdapat beberapa hal yang perlu dikembangkan pada penelitian selanjutnya, yaitu:

1. pemilihan operator genetika lain untuk memperoleh solusi pewarnaan titik minimum suatu graf.
2. penyelesaian masalah pewarnaan sisi dan daerah menggunakan algoritma genetika
3. pembandingan algoritma genetika dengan metode heuristik lain dalam menentukan solusi pewarnaan titik suatu graf, seperti algoritma semut, algoritma seleksi klonal, atau algoritma *Particle swarm optimization* (PSO).

DAFTAR PUSTAKA

Chartrand, G., dan P. Zhang. 2005. ***Introduction to Graph Theory***. New York: Mc Graw Hill Inc.

Fadlisyah, Arnawan, dan Faisal. 2009. ***Algoritma Genetika***. Graha Ilmu. Yogyakarta.

Harju, T. 2007. ***Lecture Notes on Graph Theory***. Departement of Mathematics, University of Turku, Finland.

Kokosiński, Z., Kolodziej, M., dan K. Kwarciancy. 2008. ***Parallel Genetic Algorithm for Graph Coloring Problem***. Faculty of Electrical and Computer Eng., Cracow University of Technology, Poland.

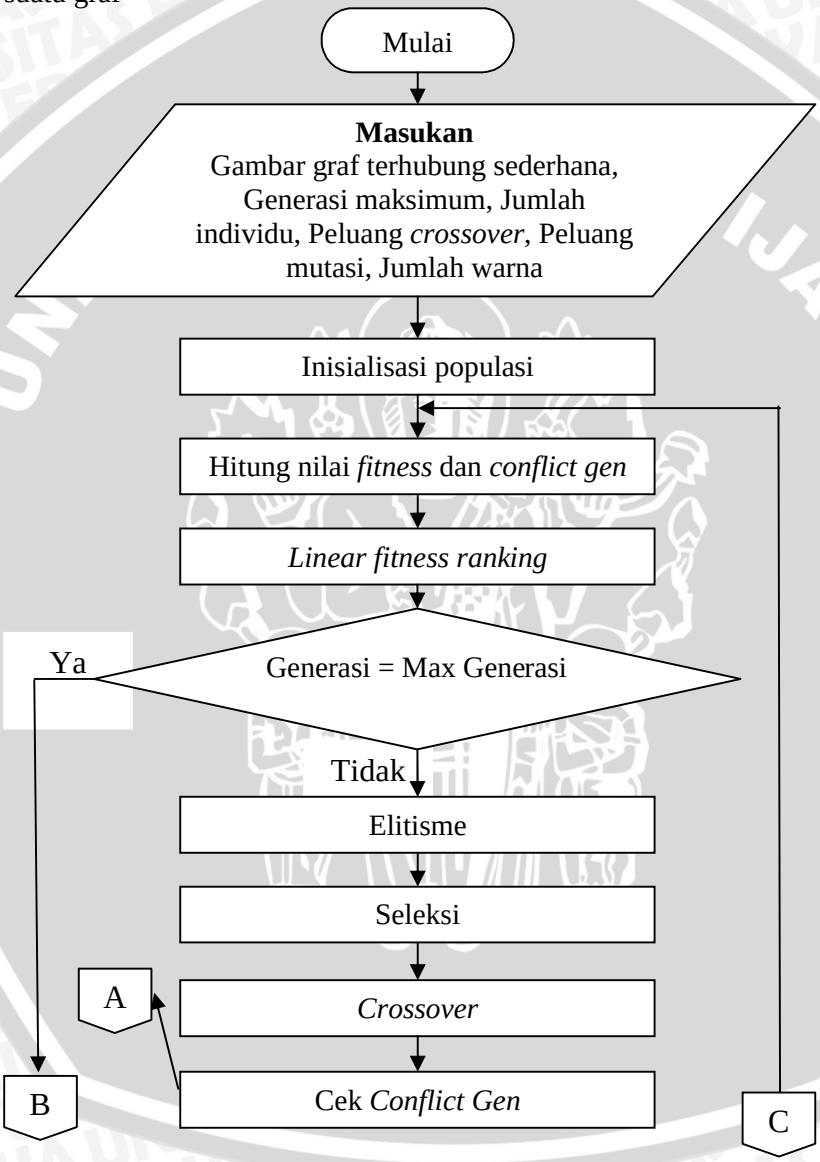
Marsudi. 1998. ***Pengantar Teori Graph***. Departemen Matematika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Brawijaya, Malang.

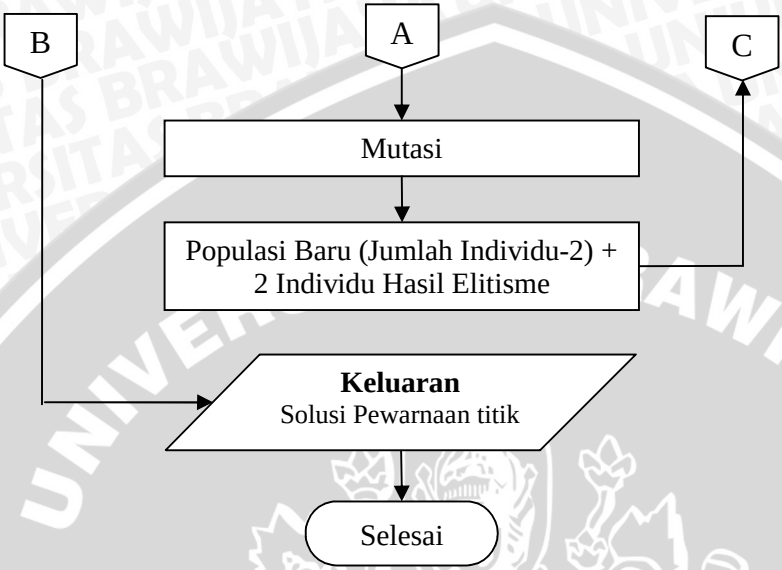
Rosen, K. H. 2003. ***Discrete Mathematics and Its Applications***. McGraw-Hill Companies, Inc. New York.

Wilson, R.J., dan J.J. Watkins. 1990. ***Graph An Introductory Approach a First Course In Discrete Mathematics***. Canada: John Willy and Sons. Inc.

LAMPIRAN

Lampiran A: Diagram alir algoritma genetika pada pewarnaan titik suatu graf




B
A
C
Mutasi
Populasi Baru (Jumlah Individu-2) +
2 Individu Hasil Elitisme
Keluaran
Solusi Pewarnaan titik
Selesai

Lampiran B: *Source code* program algoritma genetika

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms,
  Dialogs, StdCtrls, Grids, ExtCtrls, Buttons,
jpeg, Menus, OleServer,
  PowerPointXP, ComCtrls;

type
  Tgaris = record
    x,y,data1:integer;
    end;
  Tvertext = record
    x,y,Novertex:integer;
    end;
  TForm_menu = class(TForm)
    Image1: TImage;
    Panel1: TPanel;
    Panel2: TPanel;
    StringGrid1: TStringGrid;
    Panel3: TPanel;
    Image2: TImage;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    GroupBox2: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label6: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    BitBtn1: TBitBtn;
    RadioButton3: TRadioButton;
    Edit5: TEdit;
    Label7: TLabel;
```

```
    Label10: TLabel;
    MainMenu1: TMainMenu;
    Menu1: TMenuItem;
    Matriksadjency1: TMenuItem;
    Pewarnaangraf1: TMenuItem;
    AlgoritmaGenetika1: TMenuItem;
    Clear1: TMenuItem;
    Clear2: TMenuItem;
    About1: TMenuItem;
    Coloringgraph1: TMenuItem;
    GenetikAlgorithm1: TMenuItem;
    exit1: TMenuItem;
    BitBtn2: TBitBtn;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    StringGrid2: TStringGrid;
    StringGrid3: TStringGrid;
    TabSheet3: TTabSheet;
    ListBox1: TListBox;
    RetryProsesGenetika1: TMenuItem;
    TabSheet4: TTabSheet;
    ListBox3: TListBox;
    ListBox2: TListBox;
    Edit6: TEdit;
    Label4: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure exit1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var
Action: TCloseAction);
    procedure Image2MouseDown(Sender: TObject;
Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure Clear2Click(Sender: TObject);
    procedure Matriksadjency1Click(Sender:
TObject);
    procedure AlgoritmaGenetika1Click(Sender:
TObject);
    procedure RadioButton3Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure Coloringgraph1Click(Sender: TObject);
```

```
    procedure GenetikAlgorithm1Click(Sender:
TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure ListBox3Click(Sender: TObject);
    procedure T2 ;
    procedure T1;
    procedure RetryProsesGenetika1Click(Sender:
TObject);
  private
    bdg:tbitmap;
    pop_size,jvertex,jgaris,l1,l2,pil,numb:integer;
    pvertex:array[1..100] of TRect;
    pvertexh:array[1..100] of Tvertext;
    pgaris,cgaris:array[1..500] of Tgaris;
    C:array [1..50] of TColor;
    procedure lingkaran(bidang:tbitmap;
x,y:integer; hrf:string; warna:tcolor);
    procedure buatgaris(bidang:tbitmap;
x1,y1,x2,y2:integer; warna:tcolor);
    { Private declarations }
  public
    { Public declarations }
  end;

  recKROMOSOM=record
    KROMINT,KROMBINT : array [1..100] of integer;
    TOTCONF,TOTCOL : integer;
    FITNESS : real;
    STATUS  : boolean;
  end;

var
  Form_menu: TForm_menu;
  Slama:string;
  awal,akhir:tdatetime;

KROMA,KROMB,TEMPREC,KROMELITA,KROMCROSS,OFFSRING,KR
OMDOUBLE,SOLUSI,SOLTAMPIL: array [1..1000] of
recKROMOSOM;
  A,B:array[1..1000,1..1000] of integer;
  URUTAN,SELCROSS,SELUNCROSS: array[1..1000] of
integer;
```

```
  PROBSEL,PROBKUM,RANDSEL,PROBCROSS,PMUT :array
[1..1000] of real;

h,i,j,k,l,d,pil,temp,numb,gen,totMP,cr,t,maxrand,to
ttampil: integer;
  lama,tmp,totfitness,pc,pm: real;
  warna:string[20];
  state,state2 : boolean;

implementation
uses unit1, Math, StrUtils, Unit3, Unit4;
{$R *.dfm}

procedure TForm_menu.T1;
begin
       awal:=time;
end;

procedure TForm_menu.T2;
begin
  akhir:=time;
  lama:=(akhir-awal)*100000;
  Str(lama:12:12,Slama);
  edit6.Text:=Slama;
end;

procedure TForm_menu.FormCreate(Sender: TObject);
begin
  ListBox1.Items.Add(' Solusi          Fitness
n(conflict)    n(color)');
  ListBox2.Items.Add('    vertex
Color');
  bdg:=tbitmap.Create;
  bdg.Width:=image2.Width;
  bdg.Height:=Image2.Height;
  bdg.Canvas.Brush.Color:=clblack;
  bdg.Canvas.FloodFill(10,10,clBlack,fsBorder);
  Image2.Canvas.Draw(0,0,bdg);
  jvertex:=1;
  jgaris:=1;
end;
```

```
procedure TForm_menu.exit1Click(Sender: TObject);
begin
  Application.MessageBox('thanks','m4th O5',MB_OK);
  Application.Terminate;
end;

procedure TForm_menu.FormClose(Sender: TObject; var
Action: TCloseAction);
begin
  bdg.Free;
end;

procedure TForm_menu.lingkaran(bidang:tbitmap;
x,y:integer; hrf:string; warna:tcolor);
var
  lb,tg:integer;
begin
  bidang.Canvas.Pen.Color:=warna;
  bidang.Canvas.Font.Color:=warna;
  lb:=bidang.Canvas.TextWidth(hrf);
  tg:=bidang.Canvas.TextHeight(hrf);
  bidang.Canvas.Ellipse(x-10,y-10,x+10,y+10);
  bidang.Canvas.TextOut(x-(lb div 2),y-(tg div
2),hrf);
end;

procedure TForm_menu.buatgaris(bidang:tbitmap;
x1,y1,x2,y2:integer; warna:tcolor);
var
  j:TPenMode;
begin
  j:=bidang.Canvas.Pen.Mode;
  bidang.Canvas.Pen.Mode:=pmXor;
  bidang.Canvas.Pen.Color:=warna;
  bidang.Canvas.MoveTo(x1,y1);
  bidang.Canvas.LineTo(x2,y2);
  bidang.Canvas.Pen.Mode:=j;
end;

procedure TForm_menu.Image2MouseDown(Sender:
TObject; Button: TMouseButton; Shift: TShiftState;
X, Y: Integer);
```

```
var
  rgn:HRGN;
begin
  if RadioButton1.Checked then
  begin
    PVertexH[jVertex].x := X;
    PVertexH[JVertex].y := Y;
    PVertexH[JVertex].NoVertex := JVertex;
    Lingkaran(Bdg,x,y,IntToStr(jVertex),clWhite);
    PVertex[jVertex].TopLeft := Point(x-10,y-10);
    PVertex[jVertex].BottomRight :=
Point(x+10,y+10);
    Image2.Canvas.Draw(0,0,Bdg);
    inc(jVertex);
  end
  else

    //Mendeteksi Posisi Kursor pada saat ditekan
    For i := 1 to jVertex do
    begin
      rgn :=
CreateEllipticRgn(PVertex[i].Left,PVertex[i].Top,PV
ertex[i].Right,PVertex[i].Bottom);
      If (PtInRegion(rgn,x,y)) then
      begin
        if L1<>0 then
        begin
          L2 := i;
          If L2=L1 then exit;
          PGaris[jGaris].x := L1;
          PGaris[jGaris].y := L2;
          //Gambar Garis
          if L1<L2 then

BuatGaris(Bdg,PVertex[L1].Left+10,PVertex[L1].Top+1
0, PVertex[L2].Left+10,PVertex[L2].Top+10,clWhite)
          else

BuatGaris(Bdg,PVertex[L1].Left+10,PVertex[L1].Top+1
0, PVertex[L2].Left+10,PVertex[L2].Top+10,clWhite);

          //Gambar Ulang Lingkaran
```

```
Lingkaran(Bdg,PVertex[L1].left+10,PVertex[L1].Top+1
0,IntToStr(L1),clWhite);

Lingkaran(Bdg,PVertex[L2].left+10,PVertex[L2].Top+1
0,IntToStr(L2),clWhite);
          Image2.Canvas.Draw(0,0,Bdg);

          //Mengurutkan data
          k := 1;
          For j := 1 to jVertex-1 do
            For l := 1 to jGaris do
              if PGaris[l].y=j then
              begin
                CGaris[k].x := PGaris[l].x;
                CGaris[k].y := PGaris[l].y;
                CGaris[k].Data1 := PGaris[l].Data1;
                inc(k);
              end;
          PGaris := CGaris;
          k := 1;
          For j := 1 to jVertex-1 do
            For l := 1 to jGaris do
              if PGaris[l].x=j then
              begin
                CGaris[k].x := PGaris[l].x;
                CGaris[k].y := PGaris[l].y;
                CGaris[k].Data1 := PGaris[l].Data1;
                inc(k);
              end;
          PGaris := CGaris;
          L1 := 0;
          inc(jGaris);
        end
        else
          begin
            L1 := i;

Lingkaran(Bdg,PVertex[i].left+10,PVertex[i].Top+10,
IntToStr(i),clWhite);
            Image2.Canvas.Draw(0,0,Bdg);
          end;
```

```
        break;
      end;
    end;
end;

//clear graf
procedure TForm_menu.Clear2Click(Sender: TObject);
var
  kotak:Trect;
begin
  label4.Visible:=false;
  edit6.Visible:=false;
  RetryProsesGenetika1.Enabled:=true;
  bdg.Canvas.Brush.Color:=clBlack;
  kotak.TopLeft:=Point(0,0);
  kotak.BottomRight:=Point(image2.Width,
image2.Height);
  bdg.Canvas.FillRect(kotak);
  image2.Canvas.Draw(0,0,bdg);
  for i:=1 to jgaris-1 do
    for j:=1 to jvertex-1 do
      A[i,j]:=0;
  for i:=1 to jgaris-1 do
    for j:=1 to jvertex-1 do
      StringGrid1.Cells[j,i]:='';
  for i:=1 to pop_size do
  begin
    KROMA[i].TOTCONF:=0;
    KROMA[i].FITNESS:=0;
    KROMA[i].TOTCOL:=0;
    SOLTAMPIL[i].TOTCONF:=0;
    SOLTAMPIL[i].FITNESS:=0;
    SOLTAMPIL[i].TOTCOL:=0;
    SOLUSI[i].TOTCONF:=0;
    SOLUSI[i].FITNESS:=0;
    SOLUSI[i].TOTCOL:=0;
    for j:=1 to jvertex-1 do
    begin
      B[i,j]:=0;
      KROMA[i].KROMINT[j]:=0;
      KROMA[i].KROMBINT[j]:=0;
      SOLTAMPIL[i].KROMINT[j]:=0;
```

```
        SOLUSI[i].KROMINT[j]:=0;
      end;
    end;
    for i:=1 to pop_size do
      for j:=1 to jvertex+2 do
      begin
        StringGrid2.Cells[j,i]:=' ';
        StringGrid3.Cells[j,i]:=' ';
      end;
    ListBox1.Clear;
    ListBox1.Items.Add(' Kromosom          Fitness
n(conflict)    n(color)');
    ListBox3.Clear;
    ListBox3.Items.Add(' Solusi          Fitness
n(conflict)    n(color)');
    ListBox2.Clear;
    ListBox2.Items.Add('    vertex
Color');
    PageControl1.Visible:=false;
    StringGrid2.Visible:=false;
    stringgrid3.Visible:=false;
    jvertex:=1;
    jgaris:=1;
    BitBtn2.Visible:=false;
    panel2.Visible:=false;
    StringGrid1.Visible:=false;
    ListBox1.Visible:=false;
    ListBox2.Visible:=false;
    ListBox3.Visible:=false;
    GroupBox2.Visible:=false;
    RadioButton1.Checked:=false;
    RadioButton2.Checked:=false;
    RadioButton1.Enabled:=true;
    RadioButton2.Enabled:=true;
    RadioButton3.Checked:=false;
    Pewarnaangraf1.Enabled:=false;
    label6.Visible:=false;
    edit3.Visible:=false;
    label7.Visible:=false;
    label3.Visible:=false;
    edit4.Visible:=false;
    label10.Visible:=false;
```

```
  edit5.Visible:=false;
  label2.Visible:=false;
  edit2.Visible:=false;
  edit1.Clear;
  edit2.Clear;
  edit3.Clear;
  edit4.Clear;
  edit5.Clear;
end;

//matriks adjacent
procedure TForm_menu.Matriksadjency1Click(Sender:
TObject);
begin
  if jvertex-1<3 then
  begin
    Application.MessageBox('jumlah verteks tidak
dapat diproses','pesan',MB_ICONINFORMATION);
  end
  else
  begin
    Pewarnaangraf1.Enabled:=true;
    RadioButton1.Checked:=false;
    RadioButton2.Checked:=false;
    RadioButton1.Enabled:=false;
    RadioButton2.Enabled:=false;
    Panel2.Visible:=true;
    StringGrid1.Visible:=true;
    StringGrid1.ColCount:=jvertex;
    StringGrid1.RowCount:=jvertex;
    for i:=1 to jvertex-1 do
      for j:=1 to jvertex-1 do
      begin
        StringGrid1.Cells[i,j]:='';
        A[i,j]:=0
      end;
    for i:=1 to jvertex-1 do
    begin
      StringGrid1.Cells[0,i]:='v'+IntToStr(i);
      StringGrid1.Cells[i,0]:='v'+IntToStr(i);
    end;
    for i:=1 to jgaris-1 do
```

```
    begin
      A[cgaris[i].x,cgaris[i].y]:=1;
      A[cgaris[i].y,cgaris[i].x]:=1;
    end;
    for i:=1 to jvertex-1 do
      for j:=1 to jvertex-1 do
        if A[i,j]<>1 then
          A[i,j]:=0;
    for i:=1 to jvertex-1 do
      for j:=1 to jvertex-1 do
        StringGrid1.Cells[i,j]:=IntToStr(A[j,i]);
  end;
end;

procedure
TForm_menu.AlgoritmaGenetika1Click(Sender:
TObject);
begin
  RetryProsesGenetika1.Enabled:=true;
  GroupBox2.Visible:=true;
end;

procedure TForm_menu.RadioButton3Click(Sender:
TObject);
begin
  Edit5.Visible:=true;
end;

//genetika program
procedure TForm_menu.BitBtn1Click(Sender: TObject);
begin

  //pembangkitan individu acak
  if (edit1.Text=' ') then
    Application.MessageBox('lengkapi inputan
anda','pesan',MB_ICONINFORMATION)
  else
  begin
    for i:=1 to pop_size do
      for j:=1 to jvertex-1 do
        B[i,j]:=0;
    for i:=1 to pop_size do
```

```
      for j:=1 to jvertex+2 do
        StringGrid2.Cells[j,i]:=' ';
    PageControl1.Visible:=true;
    TabSheet1.Show;
    StringGrid2.Visible:=true;
    BitBtn2.Visible:=true;
    Label2.Visible:=true;
    edit2.Visible:=true;
    label6.Visible:=true;
    edit3.Visible:=true;
    label7.Visible:=true;
    label3.Visible:=true;
    edit4.Visible:=true;
    label10.Visible:=true;
    pop_size:=strtoint(edit1.Text);
    StringGrid2.RowCount:=pop_size+1;
    StringGrid2.ColCount:=jvertex;
    StringGrid3.RowCount:=pop_size+1;
    StringGrid3.ColCount:=jvertex+2;
    for i:=1 to pop_size do
    begin
      StringGrid2.Cells[0,i]:='k('+inttostr(i)+')';
      StringGrid3.Cells[0,i]:='k('+inttostr(i)+')';
    end;
    for i:=1 to jvertex-1 do
    begin
      StringGrid2.Cells[i,0]:='v('+inttostr(i)+')';
      StringGrid3.Cells[i,0]:='v('+inttostr(i)+')';
      StringGrid2.Cells[jvertex,0]:='fitness';

StringGrid2.cells[jvertex+1,0]:='tot(conflict)';
      StringGrid3.Cells[jvertex,0]:='fitness';

StringGrid3.cells[jvertex+1,0]:='tot(conflict)';
    end;
  end;

  //pengisian nilai warna pada masing-masing
kromosom
  if edit5.Text='' then
  begin
    pil:=0;
```

```
    for i:=1 to pop_size do
      for j:=1 to jvertex-1 do
      begin
        repeat
          temp:=random(jvertex);
        until(temp <> 0);
        B[i,j]:=temp;
        StringGrid2.Cells[j,i]:=intToStr(B[i,j]);
      end;
  end
  else
  begin
    pil:=1;
    numb:=StrToInt(edit5.Text);
    if (numb<3) or (numb>10) then
    begin
      Application.MessageBox('warna yang input
melebihi batas','pesan',MB_ICONINFORMATION);
      Edit5.Clear;
    end
    else
    begin
      for i:=1 to pop_size do
        for j:=1 to jvertex-1 do
        begin
          repeat
            temp:=random(numb+1);
          until(temp <> 0);
          B[i,j]:=temp;
          StringGrid2.Cells[j,i]:=intToStr(B[i,j]);
        end;
    end;
  end;
end;

procedure TForm_menu.Coloringgraph1Click(Sender:
TObject);
begin
  Form_menu.Hide;
  form3.show;
end;
```

```
procedure TForm_menu.GenetikAlgorithm1Click(Sender:
TObject);
begin
  Form_menu.Hide;
  form4.show;
end;

//proses generasi
procedure TForm_menu.BitBtn2Click(Sender: TObject);
begin
  label4.Visible:=true;
  edit6.Visible:=true;
  T1;
  c[1]:=clRed;
  c[2]:=clBlue;
  c[3]:=clGreen;
  c[4]:=clYellow;
  c[5]:=clpurple;
  c[6]:=clTeal;
  c[7]:=clFuchsia;
  c[8]:=clWhite;
  c[9]:=clLime;
  c[10]:=clAqua;

  if (edit2.Text='') or (edit3.Text='') or
(edit4.Text='') then
    Application.MessageBox('lengkapi inputan
anda','pesan',MB_ICONINFORMATION)
  else
  begin
    gen:=strtoint(edit2.Text);
    pc:=strtofloat(edit3.Text);
    pm:=strtofloat(edit4.Text);
    if ((pc>=0) and (pc<=100)) and ((pm>=0) and
(pm<=100)) then
    begin
      For i := 1 to jVertex-1 do
      begin

Lingkaran(Bdg,PVertex[i].left+10,PVertex[i].Top+10,
IntToStr(i),clWhite);
        Image2.Canvas.Draw(0,0,Bdg);
```

```
      end;
      ListBox1.Clear;
      ListBox1.Items.Add('  Kromosom        Fitness
n(conflict)    n(color)');
      ListBox3.Clear;
      ListBox3.Items.Add('  Solusi        Fitness
n(conflict)    n(color)');
      ListBox2.Clear;
      ListBox2.Items.Add('    vertex
Color');
      for i:=1 to pop_size do
      begin
        KROMA[i].TOTCONF:=0;
        KROMA[i].FITNESS:=0;
        KROMA[i].TOTCOL:=0;
        SOLUSI[i].TOTCONF:=0;
        SOLUSI[i].FITNESS:=0;
        SOLUSI[i].TOTCOL:=0;
        SOLTAMPIL[i].TOTCONF:=0;
        SOLTAMPIL[i].FITNESS:=0;
        SOLTAMPIL[i].TOTCOL:=0;
        for j:=1 to jvertex-1 do
        begin
          KROMA[i].KROMINT[j]:=0;
          KROMA[i].KROMBINT[j]:=0;
          SOLTAMPIL[i].KROMINT[j]:=0;
          SOLUSI[i].KROMINT[j]:=0;
        end;
      end;
      for i:=1 to pop_size do
        for j:=1 to jvertex+2 do
          StringGrid3.Cells[j,i]:=' ';

      ListBox1.Visible:=true;
      StringGrid3.Visible:=true;
      ListBox1.Visible:=true;
      ListBox2.Visible:=true;
      ListBox3.Visible:=true;

      //Perhitungan dalam generasi di mulai
      for h:=1 to gen do
      begin
```

```
        if h = 1 then
        begin
          for i:=1 to pop_size do
            for j:=1 to jvertex-1 do
              KROMA[i].KROMINT[j]:=B[i,j];
        end;

        for i:=1 to pop_size do
        begin
          KROMA[i].TOTCOL:=0;
          //bandingkan warna (menentukan totcol)
          if pil=0 then
          begin
            for j:=1 to jvertex-1 do
            begin
              k:=1;
              while k <= jvertex-1do
              begin
                if KROMA[i].KROMINT[k] = j then
                begin

KROMA[i].TOTCOL:=KROMA[i].TOTCOL+1;
                  k:=jvertex;
                end
                else
                  k:=k+1;
              end;
            end;
          end
          else if pil=1 then
          begin
            for j:=1 to  numb do
            begin
              k:=1;
              while k <= jvertex-1 do
              begin
                if KROMA[i].KROMINT[k] = j then
                begin

KROMA[i].TOTCOL:=KROMA[i].TOTCOL+1;
                  k:=jvertex;
                end
```

```
                else
                  k:=k+1;
              end;
            end;
          end;

          //Inisialisasi untuk mencari nilai
fitness
          for j:=1 to jvertex-1 do
            KROMA[i].KROMBINT[j]:=0;// [0 0 0 0 0]

          //menentukan kromosom biner: gen bernilai
1 bila terdapat conflict dan bewarna merah
          //dan menentukan total penalti
          totMP:=0;
          for j:=1 to jvertex-2 do
            for k:=j+1 to jvertex-1 do
              if A[j,k] = 1 then   //ketika
titiknya terhubung
                if KROMA[i].KROMINT[j] =
KROMA[i].KROMINT[k] then
                begin
                  totMP:=totMP+2;
                  KROMA[i].KROMBINT[k]:=1;
//menentukan conflict krom [0 0 1 0 1 1]
                end;

          //Menghitung total conflict
          KROMA[i].TOTCONF:=0;
          for j:=1 to jvertex-1 do

KROMA[i].TOTCONF:=KROMA[i].TOTCONF+KROMA[i].KROMBIN
T[j];

          //menentukan general penalty
          d:=0;
          if totMP > 0 then
            d:=1;

          //untuk menentukan nilai fitness

KROMA[i].FITNESS:=1/(totMP+d+KROMA[i].TOTCOL);
```

```
        end; //AKHIR PROSES EVALUASI POPULASI

        //pengurutan pertama berdasarkan nilai
fitness
        for i:=1 to pop_size-1 do
          for j:=i+1 to pop_size do
          begin
            if KROMA[i].FITNESS < KROMA[j].FITNESS
               then
            begin
              TEMPREC[i]:=KROMA[i];
              KROMA[i]:=KROMA[j];
              KROMA[j]:=TEMPREC[i];
            end;
          end;

        //pengurutan kedua berdasarkan nilai total
konflik
        for i:=1 to pop_size-1 do
          for j:=i+1 to pop_size do
          begin
            if KROMA[i].FITNESS = KROMA[j].FITNESS
               then
              if KROMA[i].TOTCONF >
KROMA[j].TOTCONF then
              begin
                TEMPREC[i]:=KROMA[i];
                KROMA[i]:=KROMA[j];
                KROMA[j]:=TEMPREC[i];
              end;
          end;

        //PROSES ELITISME
        for i:=1 to 2 do
        begin
          KROMELITA[i]:=KROMA[i];
          for j:=1 to jvertex-1 do
          begin
            KROMA[i].KROMINT[j]:=0;
            KROMA[i].KROMBINT[j]:=0;
          end;
          KROMA[i].TOTCONF:=0;
```

```
        KROMA[i].TOTCOL:=0;
        KROMA[i].FITNESS:=0;
      end;

      //pemindahan kromosom urutan ketiga menjadi
kesatu begitu seterusnya...
      for i:=3 to pop_size do
        KROMA[i-2]:=KROMA[i];

      for i:=pop_size-1 to pop_size do
      begin
        for j:=1 to jvertex-1 do
        begin
          KROMA[i].KROMINT[j]:=0;
          KROMA[i].KROMBINT[j]:=0;
        end;
        KROMA[i].TOTCONF:=0;
        KROMA[i].TOTCOL:=0;
        KROMA[i].FITNESS:=0;
      end;

      //proses seleksi
      //PROSES PENJUMLAHAN FITNESS SELEKSI
      totfitness:=0;
      for i:=1 to pop_size-2 do
        totfitness:=totfitness+KROMA[i].FITNESS;

      //Perhitungan probabilitas seleksi
      for i:=1 to pop_size-2 do
        PROBSEL[i]:=KROMA[i].FITNESS/totfitness;

      //Perhitungan probabilitas komutatif
      for i:=1 to pop_size-2 do
        PROBKUM[i]:=0;
      for i:=1 to pop_size-2 do
        for j:=1 to i do
          PROBKUM[i]:=PROBKUM[i]+PROBSEL[j];

      //PEMUTARAN RODA ROULETTE
      for i:=1 to pop_size-2 do
      begin
        RANDSEL[i]:=random;
```

```
          for j:=1 to pop_size-2 do
            if (RANDSEL[i]>PROBKUM[j-1]) and
(RANDSEL[i]<=PROBKUM[j]) then
              URUTAN[i]:=j;
        end;

        //POPULASI DOUBLE AKSEN
        for i:=1 to pop_size-2 do
          KROMB[i]:=KROMA[URUTAN[i]];

        //CROSSOVER
        k:=1;
        cr:=1;
        for i:=1 to pop_size-2 do
        begin
          PROBCROSS[i]:=random;
          if PROBCROSS[i] <= pc/100 then
          begin
            SELCROSS[k]:=i;
            k:=k+1;
          end
          else
          begin
            SELUNCROSS[cr]:=i;
            cr:=cr+1;
          end;
        end;
        k:=k-1;
        cr:=cr-1;

        //PROSESS CROSSOVER
        for i:=1 to k do
        begin

KROMCROSS[i].KROMINT:=KROMB[SELCROSS[i]].KROMINT;

KROMCROSS[i].KROMBINT:=KROMB[SELCROSS[i]].KROMBINT;
        end;

        if k>1 then
        begin
          if k mod 2 <> 0 then
```

```
          begin   //ketika jumlah orang tua ganjil
            i:=1;
            while (i<=k-2) do
            begin
              //penukaran gen orang tua ganjil
              for j:=1 to jvertex-1 do
              begin
                if KROMCROSS[i].KROMBINT[j] = 1
                  then

OFFSRING[i].KROMINT[j]:=KROMCROSS[i+1].KROMINT[j]
                else

OFFSRING[i].KROMINT[j]:=KROMCROSS[i].KROMINT[j];
              end;
              //penukaran gen orang tua genap
              for j:=1 to jvertex-1 do
              begin
                if KROMCROSS[i+1].KROMBINT[j] = 1
                  then

OFFSRING[i+1].KROMINT[j]:=KROMCROSS[i].KROMINT[j]
                else

OFFSRING[i+1].KROMINT[j]:=KROMCROSS[i+1].KROMINT[j]
              end;
              i:=i+2;
              //untuk gen orang tua terakhir yang
tidak punya pasangan
              if i = k then

OFFSRING[i].KROMINT[j]:=KROMCROSS[i].KROMINT[j];
            end;
          end
          else //untuk jumlah orang tua genap
          begin
            i:=1;
            while (i<=k-1) do
            begin
              //penukaran gen orang tua ganjil
              for j:=1 to jvertex-1 do
              begin
```

```
                if KROMCROSS[i].KROMBINT[j] = 1
                    then

OFFSRING[i].KROMINT[j]:=KROMCROSS[i+1].KROMINT[j]
                else

OFFSRING[i].KROMINT[j]:=KROMCROSS[i].KROMINT[j];
              end;
              //penukaran gen orang tua genap
              for j:=1 to jvertex-1 do
              begin
                if KROMCROSS[i+1].KROMBINT[j] = 1
                    then

OFFSRING[i+1].KROMINT[j]:=KROMCROSS[i].KROMINT[j]
                else

OFFSRING[i+1].KROMINT[j]:=KROMCROSS[i+1].KROMINT[j]
              end;
              i:=i+2;
            end;
          end;
          //Akhir proses crossover

          //penyisipan populasi yang tidak di
crossover ke populasi double aksen
          for i:=1 to cr do

KROMDOUBLE[i].KROMINT:=KROMB[SELUNCROSS[i]].KROMINT

          //setelah penyisipan diatas, disisipkan
lagi populasi hasil crossover
          for i:=cr+1 to cr+k do
            KROMDOUBLE[i].KROMINT:=OFFSRING[i-
cr].KROMINT;
        end
        else  //jika populasi kromosom hanya satu
yang di crossoverkan atau tidak ada sama sekali
        begin
          for i:=1 to pop_size-2 do
            KROMDOUBLE[i].KROMINT:=
KROMB[i].KROMINT;
```

```
        end;

        //PERHITUNGAN WARNA KONFLIK SETELAH HASIL
CROSSOVER
        for i:=1 to pop_size-2 do
        begin
          for j:=1 to jvertex-1 do
            KROMDOUBLE[i].KROMBINT[j]:=0;// [0 0 0]

          //menentukan kromosom biner: gen bernilai
1 bila terdapat konflik dan bewarna merah
          for j:=1 to jvertex-2 do
            for k:=j+1 to jvertex-1 do
              if A[j,k] = 1 then   //ketika
titiknya terhubung
                if
KROMDOUBLE[i].KROMINT[j]=KROMDOUBLE[i].KROMINT[k]
then
                  KROMDOUBLE[i].KROMBINT[k]:=1;
//menentukan conflict krom [0 0 1 0 1 1]
        end;

        //PROSES MUTASI
        for i:=1 to pop_size-2 do
          URUTAN[i]:=0;

        for i:=1 to pop_size-2 do
        begin
          PMUT[i]:=random;
          if PMUT[i]<=pm/100 then
            URUTAN[i]:=1
          else
            URUTAN[i]:=0;
        end;

        if pil = 0 then  //gunakan jvertex
          maxrand:=jvertex
        else  //gunakan numb
          maxrand:=numb;

        for i:=1 to pop_size-2 do
          if URUTAN[i]=1 then
```

```
          begin
            for j:=1 to jvertex-1 do
            begin
              if KROMDOUBLE[i].KROMBINT[j]=1 then
              begin
                repeat
                  temp:=random(maxrand);
                until(temp <> 0);
                  KROMDOUBLE[i].KROMINT[j]:=temp;
              end;
            end;
          end;
        //AKHIR DARI PROSES MUTASI, DIMANA KROMOSOM
TELAH DIPERBAHARUI

        for i:=1 to 2 do
          KROMDOUBLE[pop_size-2+i]:=KROMELITA[i];
          //KROMDOUBLE SUDAH BERISI POPULASI
KROMOSOM DAN HASIL ELITISME

        for i:=1 to pop_size do
        begin
          for j:=1 to jvertex-1 do
          begin
            KROMA[i].KROMINT[j]:=0;
            KROMA[i].KROMBINT[j]:=0;
          end;
          KROMA[i].TOTCONF:=0;
          KROMA[i].TOTCOL:=0;
          KROMA[i].FITNESS:=0;
          KROMA[i]:=KROMDOUBLE[i];
        end;
      end;//AKHIR GENERASI

      for i:=1 to pop_size do //EVALUASI POPULASI
AKHIR
      begin
        KROMA[i].TOTCOL:=0;
        //bandingkan warna (menentukan totcol)
        if pil=0 then
        begin
          for j:=1 to jvertex-1 do
```

```
  begin
    k:=1;
    while k <= jvertex-1do
    begin
      if KROMA[i].KROMINT[k] = j then
      begin
        KROMA[i].TOTCOL:=KROMA[i].TOTCOL+1;
        k:=jvertex;
      end
      else
        k:=k+1;
    end;
  end;
end
else if pil=1 then
begin
  for j:=1 to  numb do
  begin
    k:=1;
    while k <= jvertex-1 do
    begin
      if KROMA[i].KROMINT[k] = j then
      begin
        KROMA[i].TOTCOL:=KROMA[i].TOTCOL+1;
        k:=jvertex;
      end
      else
        k:=k+1;
    end;
  end;
end;

//Inisialisasi untuk mencari nilai fitness
for j:=1 to jvertex-1 do
  KROMA[i].KROMBINT[j]:=0;// [0 0 0 0 0 0]

//menentukan kromosom biner
//dan Menghitung total Penalty
totMP:=0;
for j:=1 to jvertex-2 do
  for k:=j+1 to jvertex-1 do
    if A[j,k] = 1 then
```

```
            if KROMA[i].KROMINT[j] =
KROMA[i].KROMINT[k] then
            begin
              totMP:=totMP+2;
              KROMA[i].KROMBINT[k]:=1;
//menentukan conflict krom [0 0 1 0 1 1]
            end;

        //Menghitung total conflict
        KROMA[i].TOTCONF:=0;
        for j:=1 to jvertex-1 do

KROMA[i].TOTCONF:=KROMA[i].TOTCONF+KROMA[i].KROMBIN
T[j];

        //menentukan general penalty
        d:=0;
        if totMP > 0 then
          d:=1;

        //untuk menentukan nilai fitness

KROMA[i].FITNESS:=1/(totMP+d+KROMA[i].TOTCOL);
      end; //AKHIR PROSES EVALUASI POPULASI AKHIR

      //pengurutan pertama berdasarkan fitness
      for i:=1 to pop_size-1 do
        for j:=i+1 to pop_size do
        begin
          if KROMA[i].FITNESS < KROMA[j].FITNESS
             then
          begin
            TEMPREC[i]:=KROMA[i];
            KROMA[i]:=KROMA[j];
            KROMA[j]:=TEMPREC[i];
          end;
        end;

      //pengurutan kedua berdasarkan total konflik
      for i:=1 to pop_size-1 do
        for j:=i+1 to pop_size do
        begin
```

```
          if KROMA[i].FITNESS = KROMA[j].FITNESS
             then
            if KROMA[i].TOTCONF > KROMA[j].TOTCONF
               then
            begin
              TEMPREC[i]:=KROMA[i];
              KROMA[i]:=KROMA[j];
              KROMA[j]:=TEMPREC[i];
            end;
        end;

      for i:=1 to pop_size do
      begin
        for j:=1 to jvertex-1 do

StringGrid3.Cells[j,i]:=inttostr(KROMA[i].
KROMINT[j]);

StringGrid3.Cells[jvertex,i]:=FloatToStr(KROMA[i].
FITNESS);

StringGrid3.Cells[jvertex+1,i]:=IntToStr(KROMA[i].
TOTCONF);
      end;

      SOLUSI[1]:=KROMA[1];
      i:=2;
      t:=1;
      repeat
      begin
        if KROMA[i].FITNESS = KROMA[1].FITNESS then
        begin
          if KROMA[i].TOTCONF = KROMA[1].TOTCONF
             then
          begin
            t:=t+1;
            SOLUSI[t]:=KROMA[i];
          end;
        end;
        i:=i+1;
      end;
      until (i = pop_size+1);
```

```
      for i:=1 to t do
        ListBox1.Items.Add('Kromosom
'+inttostr(i)+'
'+floattostr(SOLUSI[i].FITNESS)+'
'+inttostr(SOLUSI[i].TOTCONF)+'
'+Inttostr(SOLUSI[i].TOTCOL));

      if t = 1 then
      begin
        SOLTAMPIL[1]:=SOLUSI[1];
        ListBox3.Items.Add('Solusi '+inttostr(1)+'
'+floattostr(SOLTAMPIL[1].FITNESS)+'
'+inttostr(SOLTAMPIL[1].TOTCONF)+'
'+Inttostr(SOLTAMPIL[1].TOTCOL));
      end
      else
      begin
        tottampil:=0;
        for i:=1 to t do
        begin
          if i = 1 then
          begin
            tottampil:=tottampil+1;
            SOLTAMPIL[tottampil]:=SOLUSI[i];
          end
          else
          begin
            state2:=true;
            j:=1;
            while (j<=i-1) and (state2=true) do
            begin
              k:=1;
              state:=true;
              while (k<=jvertex-1) and (state =
true) do
              begin
                if SOLUSI[i].KROMINT[k] =
SOLUSI[j].KROMINT[k] then
                  state:=true
                else
                begin
```

```
                    state:=false;
                    k:=jvertex;
                  end;
                  k:=k+1;
                end;

                if state = true then
                  state2:=false
                else
                  state2:=true;
                j:=j+1;
              end;
              if (j=i) and (state2 = true) then
              begin
                tottampil:=tottampil+1;
                SOLTAMPIL[tottampil]:=SOLUSI[i];
              end;
            end;
          end;

          for i:=1 to tottampil do
            ListBox3.Items.Add('Solusi
'+inttostr(i)+'
'+floattostr(SOLTAMPIL[i].FITNESS)+'
'+inttostr(SOLTAMPIL[i].TOTCONF)+'
'+Inttostr(SOLTAMPIL[i].TOTCOL));
        end;
      end
      else
        begin
          Application.MessageBox('tidak dapat
diproses','ulangi inputan',MB_ICONINFORMATION);
          Edit3.Clear;
          Edit4.Clear;
        end;

  end;
  T2;
end;

procedure TForm_menu.ListBox3Click(Sender:
TObject);
```

```
begin
  i:=ListBox3.ItemIndex;
  if edit5.Text='' then
  begin
    if jvertex-1<=10 then
    begin
      ListBox2.Clear;
      ListBox2.Items.Add('    vertex       Color');
      for j:=1 to jvertex-1 do
      begin
        l:=SOLUSI[i].KROMINT[j];
        if l=1 then
          warna:='Red'
        else if l=2 then
          warna:='Blue'
        else if l=3 then
          warna:='Green'
        else if l=4 then
          warna:='Yellow'
        else if l=5 then
          warna:='Purple'
        else if l=6 then
          warna:='Teal'
        else if l=7 then
          warna:='Fuchsia'
        else if l=8 then
          warna:='White'
        else if l=9 then
          warna:='Lime'
        else
          warna:='Aqua';

Lingkaran(Bdg,Pvertex[PvertexH[j].NoVertex].Left+10
,Pvertex[PvertexH[j].NoVertex].Top+10,inttostr(Pver
texH[j].NoVertex),C[l]);
        Image2.Canvas.Draw(0,0,Bdg);
        ListBox2.Items.Add('   v +inttostr(j)+')'+'
'+warna);
      end;
    end
    else
    begin
```

```
      ListBox2.Clear;
      ListBox2.Items.Add('    vertex
Color');
      for j:=1 to jvertex-1 do
      begin
        l:=SOLUSI[i].KROMINT[j];

Lingkaran(Bdg,Pvertex[PvertexH[j].NoVertex].Left+10
,Pvertex[PvertexH[j].NoVertex].Top+10,inttostr(Pver
texH[j].NoVertex),clWhite);
        Image2.Canvas.Draw(0,0,Bdg);
        ListBox2.Items.Add('v '+inttostr(j)+')'+'
color '+inttostr(l));
      end;
    end;
  end
  else
  begin
    ListBox2.Clear;
    ListBox2.Items.Add('    vertex
Color');
    for j:=1 to jvertex-1 do
    begin
      l:=SOLUSI[i].KROMINT[j];
      if l=1 then
          warna:='Red'
        else if l=2 then
          warna:='Blue'
        else if l=3 then
          warna:='Green'
        else if l=4 then
          warna:='Yellow'
        else if l=5 then
          warna:='Purple'
        else if l=6 then
          warna:='Teal'
        else if l=7 then
          warna:='Fuchsia'
        else if l=8 then
          warna:='White'
        else if l=9 then
          warna:='Lime'
```

```
        else
          warna:='Aqua';

Lingkaran(Bdg,Pvertex[PvertexH[j].NoVertex].Left+10
,Pvertex[PvertexH[j].NoVertex].Top+10,inttostr(Pver
texH[j].NoVertex),C[l]);
      Image2.Canvas.Draw(0,0,Bdg);
      ListBox2.Items.Add('v ('+inttostr(j)+')'+'
'+warna);
    end;
  end;
end;

procedure
TForm_menu.RetryProsesGenetika1Click(Sender:
TObject);
begin
  label4.Visible:=false;
  edit6.Visible:=false;
  For i := 1 to jVertex-1 do
  begin

Lingkaran(Bdg,PVertex[i].left+10,PVertex[i].Top+10,
IntToStr(i),clWhite);
    Image2.Canvas.Draw(0,0,Bdg);
  end;
  edit1.Clear;
  edit2.Clear;
  edit3.Clear;
  edit4.Clear;
  edit5.Clear;
  edit5.Visible:=false;
  PageControl1.Visible:=false;
  StringGrid3.Visible:=false;
  ListBox1.Visible:=false;
  ListBox2.Visible:=false;
  ListBox3.Visible:=false;
  label7.Visible:=false;
  label10.Visible:=false;
  Label2.Visible:=false;
  label6.Visible:=false;
  label3.Visible:=false;
```

```
  edit2.Visible:=false;
  edit3.Visible:=false;
  edit4.Visible:=false;
  RadioButton3.Checked:=false;
  ListBox1.Clear;
  ListBox1.Items.Add('  Kromosom        Fitness
n(conflict)    n(color)');
  ListBox3.Clear;
  ListBox3.Items.Add('  Solusi        Fitness
n(conflict)    n(color)');
  ListBox2.Clear;
  ListBox2.Items.Add('    vertex
Color');
  for i:=1 to pop_size do
  begin
    KROMA[i].TOTCONF:=0;
    KROMA[i].FITNESS:=0;
    KROMA[i].TOTCOL:=0;
    SOLTAMPIL[i].TOTCONF:=0;
    SOLTAMPIL[i].FITNESS:=0;
    SOLTAMPIL[i].TOTCOL:=0;
    SOLUSI[i].TOTCONF:=0;
    SOLUSI[i].FITNESS:=0;
    SOLUSI[i].TOTCOL:=0;
    for j:=1 to jvertex-1 do
    begin
      B[i,j]:=0;
      KROMA[i].KROMINT[j]:=0;
      KROMA[i].KROMBINT[j]:=0;
      SOLTAMPIL[i].KROMINT[j]:=0;
      SOLUSI[i].KROMINT[j]:=0;
    end;
  end;
  for i:=1 to pop_size do
    for j:=1 to jvertex+2 do
    begin
      StringGrid2.Cells[j,i]:=' ';
      StringGrid3.Cells[j,i]:=' ';
    end;
end;

end.
```