

Pengantar Sistem Cerdas
Tugas Besar “Permainan Mini Sudoku Menggunakan Genetic Algorithm”



Disusun oleh:
6181901041 - Nathanael Adi Trianto
6181901054 - Rafael Hendrajat Widyadana

Program Studi Teknik Informatika
Fakultas Teknologi Informasi dan Sains
Universitas Katolik Parahyangan
Tahun 2022/2023

DAFTAR ISI

DAFTAR ISI	2
BAB I PENDAHULUAN	3
1.1 Latar Belakang	3
1.2 Rumusan Masalah	4
1.3 Tujuan dan Manfaat	4
BAB II LANDASAN TEORI	5
2.1 Aturan Dasar	5
2.2. Algoritma Genetik:	6
Sejarah Algoritma Genetik	6
Cara kerja algoritma secara detail:	6
BAB III ANALISIS MASALAH	9
3.1. Pengalaman saat bermain	9
3.2. Detail Perancangan Algoritma Genetik	10
BAB IV IMPLEMENTASI	15
BAB V EKSPERIMEN	21
BAB VI KESIMPULAN	31
4.1. Kesimpulan	31
4.2. Saran	31
4.3. Referensi	31

BAB I

PENDAHULUAN

1.1 Latar Belakang

Permainan Sudoku 4x4 adalah varian yang lebih sederhana dari permainan Sudoku klasik yang lebih terkenal. Latar belakang dari permainan ini dapat ditelusuri kembali ke permainan Latin Squares yang pertama kali diperkenalkan oleh matematikawan Swiss Leonhard Euler pada abad ke-18. Namun, permainan Sudoku dalam bentuk yang kita kenal sekarang ini berkembang lebih lanjut pada abad ke-20 di Jepang.

Sudoku menggunakan Latin Squares. Latin Squares adalah struktur matematika yang muncul dalam studi tentang permutasi. Euler mengamati hubungan antara angka dan simbol dalam matriks berukuran kecil yang diatur sedemikian rupa sehingga tidak ada simbol yang muncul lebih dari sekali di setiap baris dan kolom. Konsep inilah yang menjadi dasar bagi banyak permainan penyelesaian angka dan simbol.

Pengembangan sudoku yang terjadi di Jepang. Meskipun dasar-dasar Latin Squares sudah ada, permainan Sudoku dalam format modern dikembangkan di Jepang. Pada tahun 1979, Howard Garns, seorang perancang puzzle Amerika, menciptakan versi awal permainan ini dengan nama "Number Place". Meskipun permainan ini belum meraih popularitas yang besar pada saat itu, namun konsepnya diperkenalkan di Jepang pada awal tahun 1984 oleh penerbit puzzle bernama Nikoli. Mereka memberikan nama "Sudoku", yang berarti "angka satu per baris", sebagai merek dagang untuk permainan tersebut.

Sudoku dengan cepat menjadi sangat populer di Jepang dan kemudian menyebar ke seluruh dunia pada awal abad ke-21. Banyak surat kabar dan majalah di berbagai negara mulai menyertakan papan Sudoku di bagian puzzle mereka. Internet juga membantu dalam penyebaran global permainan ini, memungkinkan orang dari seluruh dunia bermain dan membagikan tantangan Sudoku secara online.

Sudoku dengan varian 4x4: Seiring waktu, berbagai varian Sudoku mulai muncul, salah satunya adalah Sudoku 4x4. Varian ini dirancang untuk memberikan tantangan lebih ringan dibandingkan dengan Sudoku 9x9 yang lebih kompleks. Dalam Sudoku 4x4, papan permainan terdiri dari kotak 4x4 yang harus diisi dengan angka 1 hingga 4. Tantangan utamanya adalah memastikan bahwa setiap angka muncul tepat sekali dalam setiap baris dan kolom, serta setiap blok 2x2 di papan permainan.

Kesederhanaan Sudoku 4x4 menjadikannya pilihan yang baik untuk pemula atau orang yang ingin mencoba permainan ini tanpa terlalu diperhadapkan pada kompleksitas penuh dari varian 9x9.

1.2 Rumusan Masalah

Berikut ini adalah rumusan masalah yang digunakan sebagai pedoman dalam membangun agen sistem cerdas untuk permainan Mini Sudoku 4*4 kali ini:

1. Bagaimana cara membangun agen sistem cerdas dengan *fitness function* yang cukup kecil untuk permainan Mini Sudoku 4*4 ini?
2. Bagaimana cara menyelesaikan permainan Mini Sudoku 4*4 ini dengan agen sistem cerdas yang akan dibuat?

1.3 Tujuan dan Manfaat

Dari rumusan masalah di atas, tujuan dari pembuatan agen sistem cerdas untuk permainan Mini Sudoku 4*4 ini adalah:

1. Untuk dapat menentukan *fitness function* terkecil yang dapat menyelesaikan permainan Mini Sudoku 4*4 ini dengan baik
2. Menyelesaikan permainan Mini Sudoku 4*4 dengan agen sistem cerdas yang akan dibuat.

BAB II

LANDASAN TEORI

2.1 Aturan Dasar

Sudoku 4*4 adalah varian permainan Sudoku yang lebih kecil dengan kotak berukuran 4*4 dan sub-kotak 2*2. Aturan dasar Sudoku 4*4 tetap sama dengan Sudoku tradisional, yaitu mengisi setiap sel kosong dengan angka 1 hingga 4 sedemikian rupa sehingga setiap angka hanya muncul satu kali dalam setiap baris, kolom, dan sub-kotak 2*2.

Aturan dasar pada permainan Sudoku 4x4 adalah:

1. Setiap baris harus berisi angka 1,2,3, dan 4 tanpa pengulangan.
2. Setiap kolom harus berisi angka 1,2,3, dan 4 tanpa pengulangan.
3. Setiap sub-kotak 2*2 (dalam kotak 4*4) harus berisi angka 1,2,3, dan 4 tanpa pengulangan.
4. Pada papan awal, beberapa angka mungkin sudah diisi. Pemain harus melengkapi papan sehingga mematuhi aturan-aturan di atas.

Misal, berikut adalah contoh papan Sudoku 4*4 yang belum lengkap:

1	2	3	4
3	4	1	2
2	1	4	3
4		2	

Pada contoh di atas, baris pertama dan kolom pertama telah diisi dengan 1,3,2,4. Namun, ada beberapa celah yang masih perlu diisi untuk mematuhi aturan-aturan tersebut. Tugas Pemain adalah melengkapi papan dengan mengikuti aturan di atas hingga semua kotak terisi dengan benar. Jika pemain ingin memecahkan sebuah Sudoku 4*4, pemain bisa mulai dengan melihat

baris, kolom, dan sub-kolom yang sudah memiliki beberapa angka yang terisi dan mencoba mengisi angka yang belum ada sesuai dengan aturan di atas.

2.2. Algoritma Genetik:

Sejarah Algoritma Genetik

Algoritma genetika dikembangkan oleh John Holland dari University of Michigan di tahun 1962, yang awalnya terinspirasi oleh teori evolusi. Kemudian pada tahun 1975 buku John Holland dan David Goldberg menerbitkan *Adaptasi dalam Sistem Alami dan Buatan*. Buku tersebut menceritakan bagaimana sistem cerdas bekerja berdasarkan seleksi alam dan genetika. Tujuh belas tahun kemudian, John Koza melakukan penelitian tentang program tersebut Menggunakan algoritma genetika yang dikenal dengan “Pemrograman Genetik”. Itu menggunakan LIPS, bahasa pemrograman tertua kedua setelah itu. Sejauh ini, algoritma genetika telah digunakan untuk menyelesaikannya Masalah sulit dipecahkan dengan algoritma pencarian umum

Cara kerja algoritma secara detail:

Algoritma Genetika adalah sebuah teknik pencarian dengan metode acak, hal ini dapat dilihat dari proses yang dilakukan. Terdapat beberapa komponen dasar yang ada di Algoritma genetik, antara lain:

- Fitness Function yang digunakan untuk optimasi nilai mendekati hasil
- Populasi kromosom dalam satu generasi
- Pemilihan kromosom yang ingin direproduksi (selection)
- Persilangan untuk menghasilkan kromosom yang baru (crossover)
- Mutasi kromosom dengan bilangan acak dimana biasanya memiliki kemungkinan 0,001 (mutation)

Berikut ini proses-proses pembentukan algoritma genetik.

1. langkah pertama dalam siklus hidup algoritma genetika adalah mendefinisikan ruang solusi
2. Membuat parameter pada algoritma dimana parameter tersebut adalah
 - a. Chromosome
 - b. Besaran Populasi
 - c. Inisiasi Populasi
 - d. Jumlah Keturunan
 - e. Metode pemilihan parent
 - f. Metode Crossover
 - g. Metode Mutasi
 - h. Metode Pemilihan generasi
 - i. Kondisi dimana suatu variabel harus berhenti karena merupakan nilai akhir didapatkan

3. Kemudian membuat initial populasi (setting sebuah parameter dimana nantinya akan muncul populasi yang mungkin terjadi)

contoh algoritma initial populasi

```
generate_initial_population (population_size, individual_size)
  let population be an empty array
  for individual in range 0 to population_size
    let current_individual be an empty array
    for gene in range 0 to individual_size
      let random_gene be 0 or 1 randomly
      append random_gene to current_individual
    append current_individual to population
  return population
```

4. Memperoleh nilai Fitness yang mana akan ditentukan populasi mana yang mendekati atau bahkan mungkin solusi dari algoritma Genetic tersebut.

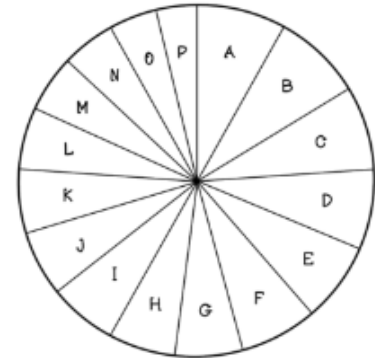
Contoh algoritma mencari nilai fitness

```
calculate_individual_fitness (individual,
                              knapsack_items,
                              knapsack_max_weight)

  let total_weight equal 0
  let total_value equal 0
  for gene_index in range 0 to length of individual
    let current_bit equal individual[gene_index]
    if current_bit equals 1
      add weight of knapsack_items[gene_index] to total_weight
      add value of knapsack_items[gene_index] to total_value
  if total_weight is greater than knapsack_max_weight
    return value as 0 since it exceeds the weight constraint
  return total_value as individual fitness
```

5. Seleksi parent yang nantinya di crossover dan dijadikan generasi yang baru. Dimana pemilihan parentnya sendiri seperti yang diilustrasikan pada gambar dibawah

A	1 0 1 1 1 0 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 0 1	13,107,019
B	1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 1 0 1 0 0 0 1 0 0 0	12,965,145
C	0 0 1 1 0 1 1 0 1 0 0 1 1 0 0 0 0 1 0 1 0 1 1 0 0 0	12,344,873
D	0 0 1 1 1 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 0 0	11,739,363
E	1 1 0 0 0 1 0 0 0 1 1 1 1 1 1 0 1 1 0 1 0 0 0 1 0 0 0	11,711,159
F	1 1 0 0 0 1 0 0 1 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0	11,611,967
G	1 0 1 0 0 1 1 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 1 0	10,042,441
H	1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 0 0 0 0 0 0	9,883,682
I	1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0	9,857,597
J	0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1	9,670,184
K	0 0 0 0 1 1 0 1 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0	9,277,580
L	1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 1 0 0 0 1 0 1 0 0	8,931,719
M	0 1 0 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0	8,324,936
N	1 1 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0	8,018,760
O	0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1	6,900,314
P	0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0	6,056,664



6. Memproduksi keturunan, ada 2 hal yang akan terjadi ketika melakukan reproduksi keturunan
 - a. Crossover : mencampur bagian kromosom dari parent
 - b. Mutasi : mengganti gene pada keturunan sehingga terjadi diversity
7. Melakukan populasi pada generasi berikutnya
8. dihitung kembali fitness dari generasi yang baru sehingga nantinya menciptakan evolusi individu yang merupakan solusi nantinya
9. Kembali ke langkah lima sampai solusi ditemukan atau berhenti sampai generasi ke sekian dimana dibatasi supaya(worst case) tidak terus menerus melakukan perhitungan yang lama

BAB III

ANALISIS MASALAH

3.1. Pengalaman saat bermain

Terdapat heuristik yang didapat saat mencoba permainan puzzle ini, diantaranya:

1. Memprioritaskan untuk memasang angka yang memungkinkan pada kolom yang kosong.
2. Langsung memasukan angka dengan kemungkinan nilai pada sub-kotak yang sudah memiliki angka pada kotaknya

2		3	
	3		
	4		3

4x4 Easy Sudoku

3		1	
	2		4

3. Menghindari pemasangan angka yang sama dengan melihat angka yang terdapat pada kolom, baris serta sub-kotak yang ada, sehingga kita dapat melakukan eliminasi apabila angka tersebut terdapat pada pada baris, kolom, ataupun sub-kotak dan tidak memasukan angka tersebut.

2	1	3	
4	3		
	2		
	4		3

3	4	1	2
1	2	3	4

3.2. Detail Perancangan Algoritma Genetik

Perancangan algoritma genetik untuk Sudoku 4x4 melibatkan langkah-langkah berikut:

1. Representasi Individu:

Setiap individu dalam populasi akan direpresentasikan sebagai sebuah matriks 4x4 yang merepresentasikan papan Sudoku. Setiap sel dalam matriks akan berisi angka dari 1 hingga 4.

Contoh representasi individu:

1	2	3	4
3	4	2	1
2	1	4	3
4	3	1	2

2. Populasi Awal:

Membuat populasi awal dengan sejumlah individu secara acak atau menggunakan teknik seeding untuk memastikan keberagaman populasi.

3. Evaluasi Fitness:

Menghitung nilai fitness untuk setiap individu dalam populasi. Fitness diukur dengan menghitung jumlah kolom, baris, dan sub-grid yang memiliki semua angka dari 1 hingga 4 tanpa pengulangan. Individu dengan lebih sedikit kesalahan memiliki fitness yang lebih tinggi.

4. Seleksi:

Memilih individu-individu dengan probabilitas yang lebih tinggi untuk menjadi orangtua dalam proses reproduksi. Seleksi dapat menggunakan teknik roulette wheel, turnamen, atau metode seleksi lainnya.

5. Crossover (Reproduksi):

Pasangkan individu yang telah dipilih untuk membuat anak-anak baru. Dalam crossover, dua individu (orangtua) menghasilkan dua anak dengan menukar sebagian matriks mereka.

Contoh crossover:

Parent 1:

1	2	3	4
3	4	2	1
2	1	4	3
4	3	1	2

Parent 2:

4	3	2	1
2	1	4	3
3	4	1	2
1	2	3	4

Anak 1 (menggunakan baris pertama dari parent 1 dan sisanya dari orangtua 2):

1	2	3	4
2	1	4	3
3	4	1	2
4	3	1	2

Anak 2 (menggunakan baris pertama dari parent 2 dan sisanya dari orangtua 1):

4	3	2	1
3	4	2	1
2	1	4	3
1	2	3	4

6. Mutasi:

Untuk menjaga keberagaman dalam populasi, beberapa individu mungkin mengalami mutasi. Mutasi dapat mengganti nilai acak dalam matriks individu.

Contoh mutasi (mengganti nilai acak):

1	2	3	4
3	4	2	1
2	1	4	3
4	3	1	2

menjadi

1	2	3	4
3	4	2	1
2	1	4	3
4	1	1	2

7. Evaluasi Generasi Baru:

Menghitung fitness dari individu-individu dalam generasi baru.

8. Kriteria Berhenti:

Mengulang langkah-langkah di atas hingga salah satu dari kriteria berhenti terpenuhi, seperti mencapai solusi optimal atau melewati jumlah iterasi maksimum.

9. Solusi:

Apabila algoritma yang kami terapkan berhasil dalam menemukan solusi, maka dapat dipastikan bahwa matriks individu terbaik yang terdapat dalam populasi adalah representasi dari solusi Sudoku 4x4 yang diinginkan. Dalam konteks ini, kesuksesan algoritma bukan hanya sekadar mencapai solusi, melainkan juga mengidentifikasi matriks individu dengan tingkat kecocokan (fitness) tertinggi yang merepresentasikan solusi yang akurat dan benar dalam permainan Sudoku 4x4 tersebut. Hasil ini menegaskan bahwa algoritma genetik telah berhasil mengatasi tantangan permainan Sudoku dan memunculkan matriks individu yang sesuai dengan aturan permainan.

10. Output:

Matriks individu terbaik sebagai solusi dari Sudoku 4x4.

Dengan mengulangi langkah-langkah di atas, algoritma genetik akan mencoba untuk menemukan solusi yang memenuhi aturan Sudoku 4x4. Setiap iterasi populasi akan berkembang lebih baik karena individu dengan fitness yang lebih tinggi akan lebih cenderung berkembang.

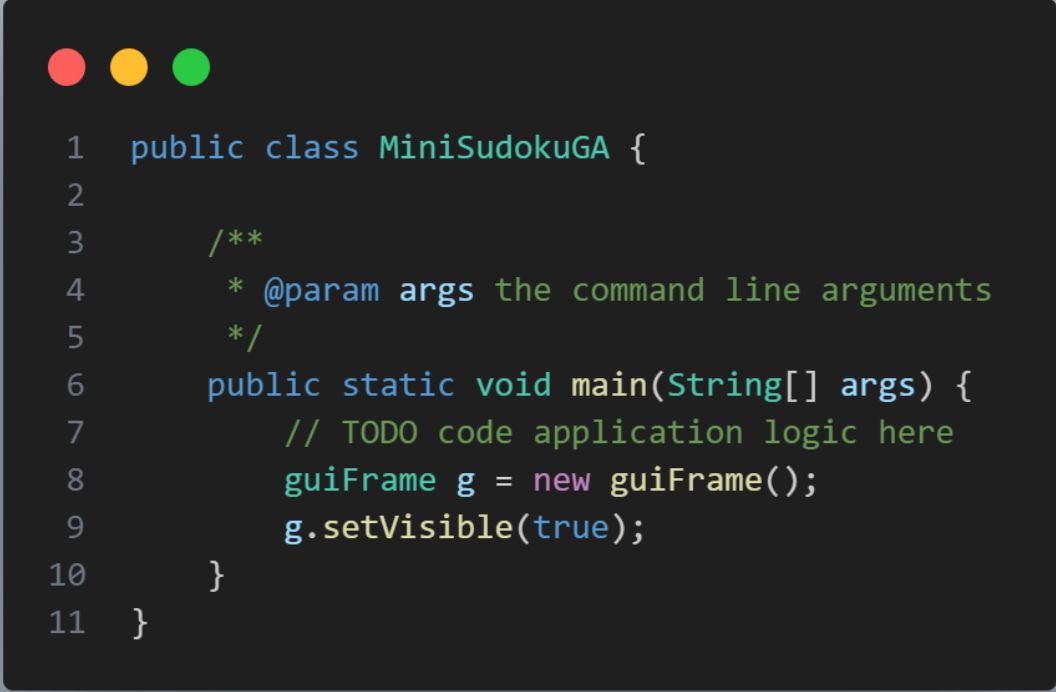
BAB IV

IMPLEMENTASI

Pada implementasi algoritma genetik untuk penyelesaian Sudoku 4x4 dibuat beberapa class antara lain class MiniSudokuGA, class gui Frame, class Chromosome, class Gene, dan class GeneticAlgorithm yang akan dijelaskan sebagai berikut:

a. class Main

class ini merupakan class utama untuk menjalankan program algoritma genetik. Berikut ini adalah gambar yang berisi potongan kode pada class Main



```
1 public class MiniSudokuGA {
2
3     /**
4      * @param args the command line arguments
5      */
6     public static void main(String[] args) {
7         // TODO code application logic here
8         guiFrame g = new guiFrame();
9         g.setVisible(true);
10    }
11 }
```

Pada dasarnya class MiniSudokuGA hanyalah class yang berisikan method main saja. Method main tersebut akan memanggil class guiFrame.

b. class guiFrame

Pada class guiFrame, terdapat method yang penting, seperti loadButton, dan juga solveButton.

Method loadButton berfungsi untuk melakukan penguploadan file berupa .txt file.

```
1 private void LoadButtonActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_LoadButtonActionPerformed
2     // TODO add your handling code here:
3     JFileChooser OpenFile = new JFileChooser();
4
5     int pFile = OpenFile.showOpenDialog(this);
6     if (pFile == JFileChooser.APPROVE_OPTION) {
7         File file = OpenFile.getSelectedFile();
8         try {
9             s = new Sudoku(4, file);
10            showBoard();
11        } catch (Exception e) {
12            System.out.println("");
13        }
14    }
15 }GEN-LAST:event_LoadButtonActionPerformed
```

Method solveButton berfungsi untuk melakukan solving terhadap file input sudoku yang sudah diupload dengan cara memanggil method iteration yang berada pada class GeneticAlgorithm.

```
1 private void SolveButtonActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_SolveButtonActionPerformed
2     // TODO add your handling code here:
3     ga = new GeneticAlgorithm(4, s.puzzle, 0.6, 0.1, 20, 25);
4     ga.iteration();
5     showBoard();
6 }GEN-LAST:event_SolveButtonActionPerformed
```

c. Implementasi metode crossover


```

1  void crossover(){
2      for (int i = 0; i < popSize; i += 2) {
3          if (Math.random() < probCO) {
4              // Pilih dua kromosom induk secara acak
5              int parent1Index = (int) (Math.random() * popSize);
6              int parent2Index;
7              do {
8                  parent2Index = (int) (Math.random() * popSize);
9              } while (parent2Index == parent1Index);
10
11             Chromosome parent1 = population[parent1Index];
12             Chromosome parent2 = population[parent2Index];
13
14             // Pilih titik pemotongan (crossover point) secara acak
15             int crossoverPoint = (int) (Math.random() * (size - 1)) + 1;
16
17             // Buat dua kromosom anak
18             Chromosome child1 = new Chromosome(size, puzzle);
19             Chromosome child2 = new Chromosome(size, puzzle);
20
21             // Lakukan crossover pada titik pemotongan
22             for (int j = 0; j < size; j++) {
23                 if (j < crossoverPoint) {
24                     child1.setGene(j, parent1.getGene(j));
25                     child2.setGene(j, parent2.getGene(j));
26                 } else {
27                     child1.setGene(j, parent2.getGene(j));
28                     child2.setGene(j, parent1.getGene(j));
29                 }
30             }
31
32             // Gantikan kromosom induk dengan anak
33             population[i] = child1;
34             population[i + 1] = child2;
35         }
36     }
37 }

```

Metode crossover adalah langkah dalam algoritma genetika yang digunakan untuk menciptakan kromosom baru dari dua kromosom yang ada. Dalam konteks permainan Sudoku 4x4 ini, metode ini digunakan untuk membuat dua kromosom anak yang akan menggantikan dua kromosom induk dalam populasi.

Langkah langkah dari metode ini yaitu:

1. Pertama-tama, kita melalui seluruh populasi kromosom dengan langkah dua kromosom pada setiap iterasinya.
2. Setiap saat, kita pertimbangkan apakah akan melakukan crossover antara dua kromosom. Ini ditentukan secara acak berdasarkan probabilitas probCO (probCO adalah singkatan dari "Probability of Crossover" atau Probabilitas Crossover dalam konteks algoritma genetika. Probabilitas ini adalah probabilitas bahwa dua kromosom dalam populasi akan mengalami operasi crossover dalam satu iterasi atau generasi dari algoritma genetika). Jika angka acak yang dihasilkan lebih kecil daripada probCO, maka crossover akan dilakukan. Ini memungkinkan variasi dalam populasi.
3. Setelah memutuskan untuk melakukan crossover, kita pilih dua kromosom induk secara acak dari populasi. Pastikan kromosom yang kita pilih berbeda satu sama lain.
4. Selanjutnya, kita pilih titik pemotongan (crossover point) dalam kromosom secara acak. Pemilihan titik pemotongan ini menentukan bagian mana dari kromosom yang akan pertukarkan antara kedua kromosom anak.
5. Kita buat dua kromosom anak baru. Kromosom anak pertama akan memiliki sebagian gen dari kromosom induk pertama dan sebagian gen dari kromosom induk kedua, sesuai dengan titik pemotongan. Kromosom anak kedua akan memiliki sebagian gen dari kromosom induk kedua dan sebagian gen dari kromosom induk pertama.
6. Terakhir, kita menggantikan dua kromosom induk dengan dua kromosom anak dalam populasi. Ini berarti kromosom anak pertama menggantikan salah satu dari kromosom induk, dan kromosom anak kedua menggantikan kromosom induk yang lain.

Setiap kali metode crossover dijalankan, metode ini menciptakan variasi dalam populasi dengan menggabungkan sifat-sifat genetik dari dua kromosom yang ada. Variasi ini membantu algoritma genetika dalam mencari solusi yang lebih baik untuk Sudoku.

d. Implementasi metode mutation

```

1  void mutation(){
2      for (int i = 0; i < popSize; i++) {
3          if (Math.random() < probM) {
4              // Pilih satu kromosom secara acak
5              Chromosome chromosome = population[i];
6
7              // Pilih satu gen dalam kromosom untuk dimutasi secara acak
8              int mutationPoint = (int) (Math.random() * size);
9
10             // Lakukan mutasi pada gen yang dipilih
11             chromosome.mutateGene(mutationPoint);
12         }
13     }
14 }

```

Implementasi mutation adalah bagian dari algoritma genetika yang bertugas untuk melakukan operasi mutasi pada populasi kromosom. Mutasi merupakan salah satu langkah penting dalam algoritma genetika yang digunakan untuk memperkenalkan variasi genetik ke dalam populasi kromosom.

Langkah-langkah dari metode ini yaitu:

1. `for (int i = 0; i < popSize; i++)` : Loop ini akan mengulangi seluruh kromosom dalam populasi.
2. `if (Math.random() < probM)` : Pada setiap iterasi, kita memutuskan apakah akan melakukan mutasi atau tidak dengan menghasilkan angka acak antara 0 dan 1 menggunakan `Math.random()`. Jika angka acak tersebut lebih kecil daripada `probM`, maka operasi mutasi akan dilakukan pada kromosom saat ini. Ini memungkinkan kita untuk mengendalikan seberapa sering mutasi terjadi.
3. `Chromosome chromosome = population[i]` : Kromosom yang akan dimutasi dipilih dari populasi berdasarkan indeks saat ini.
4. `int mutationPoint = (int) (Math.random() * size)` : Dalam kromosom yang dipilih, kita memilih satu gen secara acak untuk dimutasi. `mutationPoint` adalah indeks gen yang akan dimutasi.

5. `chromosome.mutateGene(mutationPoint)` : Mutasi dilakukan pada gen yang dipilih dalam kromosom saat ini. Metode `mutateGene(mutationPoint)` pada objek kromosom akan mengganti nilai gen tersebut dengan nilai acak atau sesuai dengan aturan mutasi yang ditetapkan.

Ketika melakukan operasi mutasi secara acak pada beberapa kromosom dalam populasi, kita dapat menjaga variasi genetik dalam populasi selama proses evolusi. Hal ini membantu mencegah terjadinya konvergensi prematur ke solusi yang kurang optimal dan meningkatkan kemungkinan menemukan solusi terbaik. Probabilitas `probM` digunakan untuk mengontrol seberapa sering mutasi akan terjadi dalam algoritma genetika. Semakin tinggi nilai `probM`, semakin sering mutasi akan terjadi.

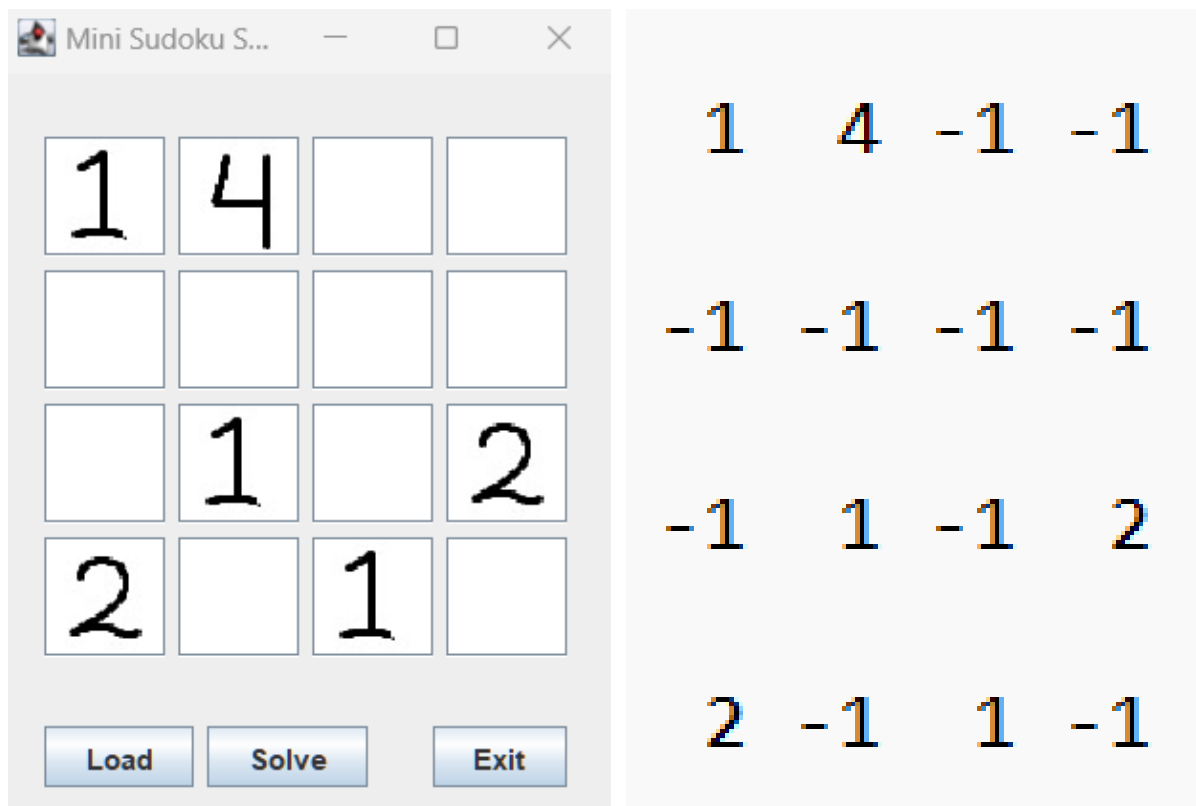
BAB V

EKSPERIMEN

Eksperimen ini dilakukan setelah kami mengimplementasikan Algoritma Genetik untuk penyelesaian permainan Sudoku 4x4 ini. Kami melakukan eksperimen pada 50 jenis test case yang berbeda dengan ukuran papan yang sama, yaitu 4×4 . Untuk hasil penyelesaian permainan, program ini akan menampilkan hasil dari fitness function di mana fitness function tersebut merupakan hasil terbaik dari permainan ini.

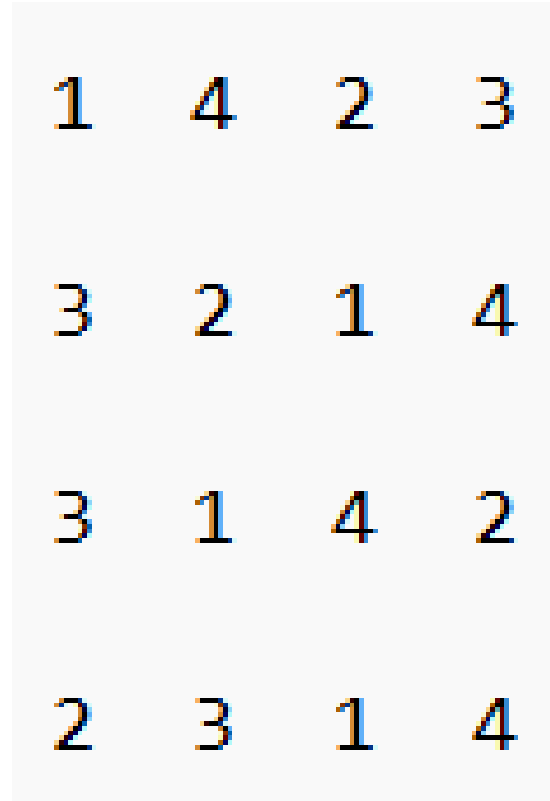
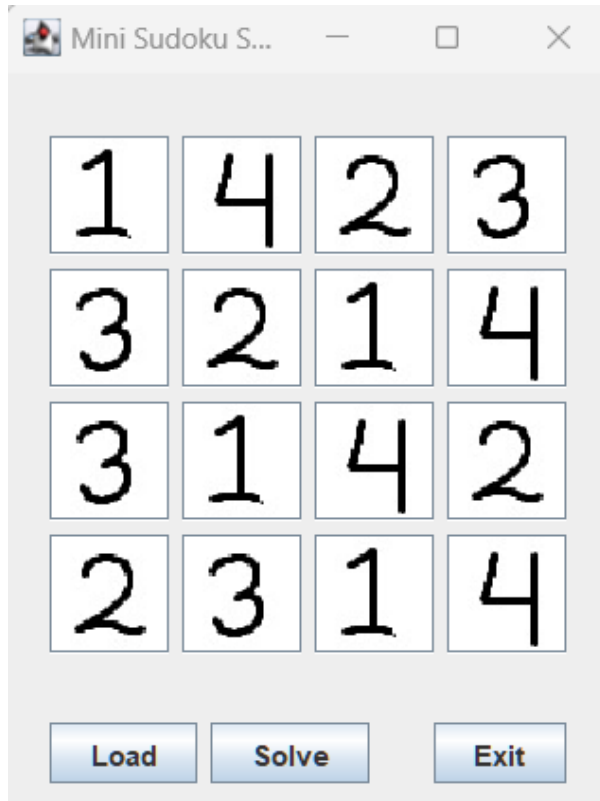
5.1 Test Case 1:

Berikut test case 1 yang kami uji menggunakan algoritma genetik yang kami buat:



Gambar Test Case 1 Beserta Encodingnya

Lalu hasil yang didapatkan oleh algoritma ini adalah sebagai berikut:



Gambar Test Case 1 yang sudah diselesaikan beserta Encodingnya

Hasil tersebut didapatkan dengan melakukan percobaan eksperimen dengan parameter generasi maksimum adalah 40. Berikut hasil keluaran yang ditampilkan oleh program:

The list is: [2, 3, 3, 2, 1, 4, 3, 4, 3, 4]

The fitness is: 5

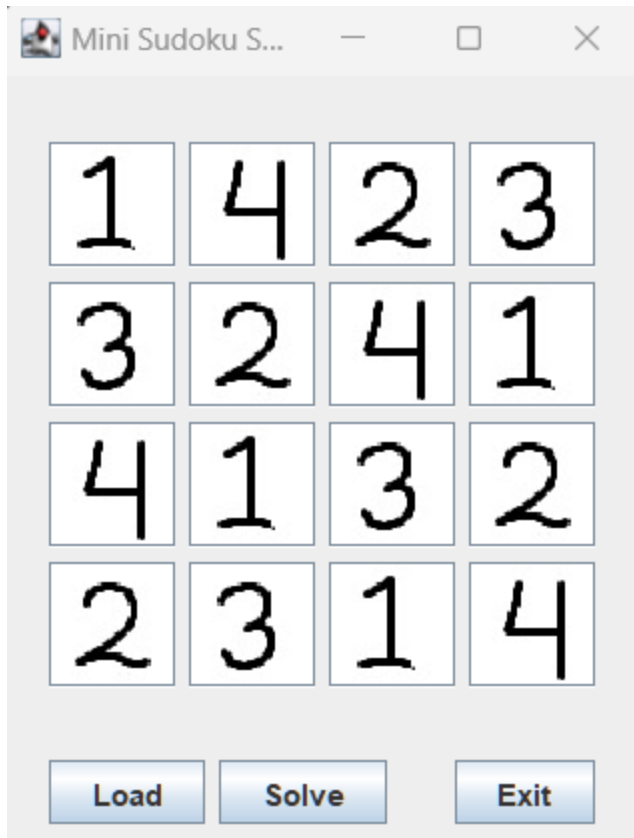
```

-----
| 1 | 4 | 2 | 3 |
-----
| 3 | 2 | 1 | 4 |
-----
| 3 | 1 | 4 | 2 |
-----
| 2 | 3 | 1 | 4 |
-----

```

Dari keluaran yang ditampilkan oleh program untuk test case yang pertama. Dapat dilihat bahwa program menemukan fitness terbaik ada pada generasi ke 30 dengan nilai fitness 5. Namun hasil dari pengujian tersebut bukanlah hasil yang diharapkan.

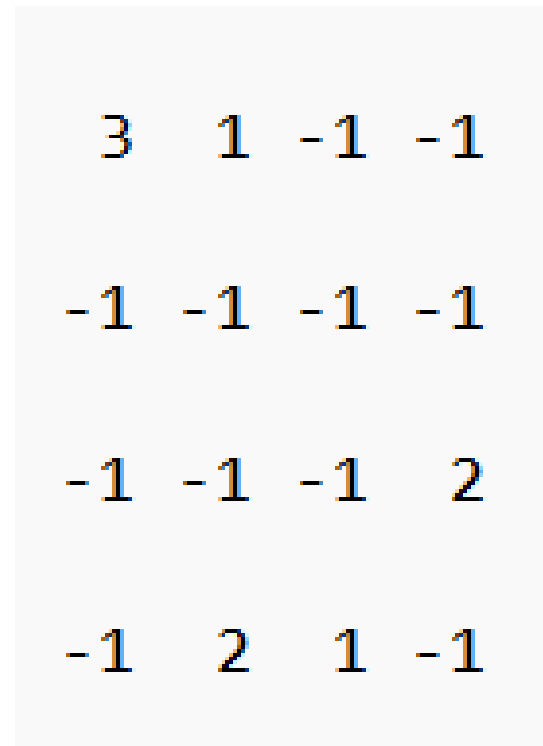
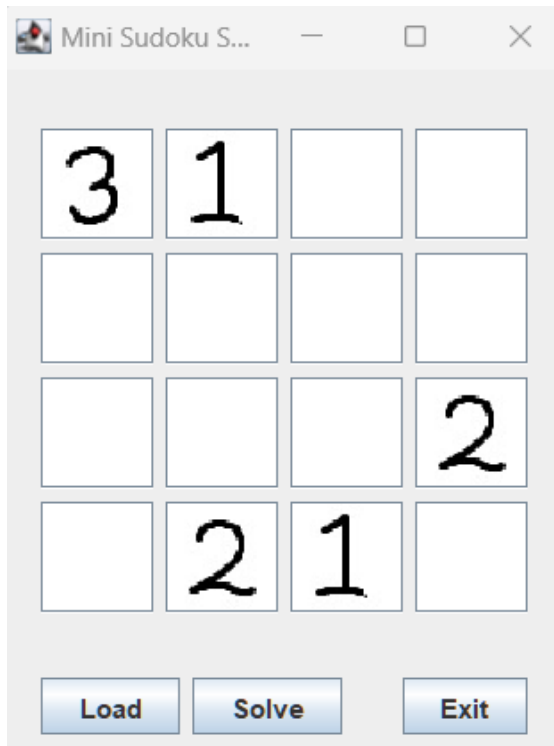
Contoh hasil yang diharapkan:



Gambar dari contoh hasil yang diharapkan

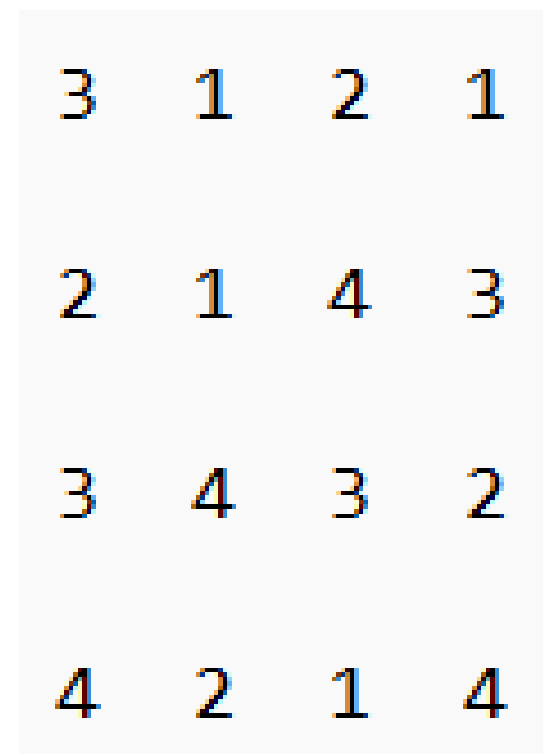
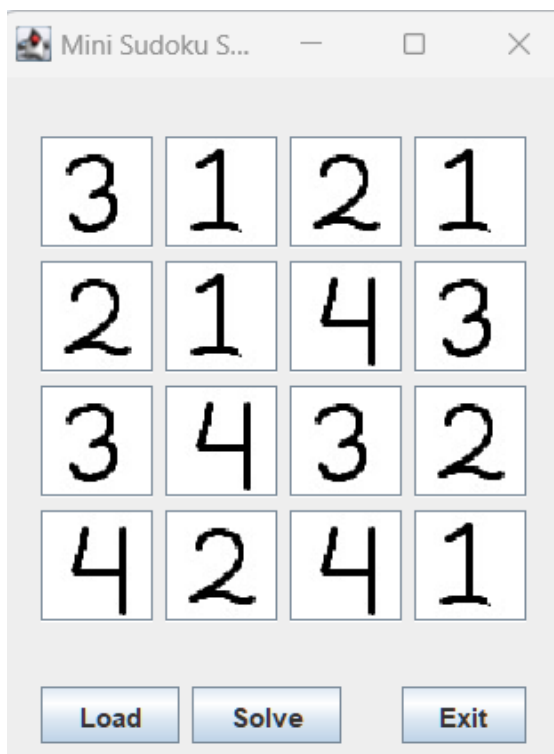
5.2 Test Case 2

Berikut test case 2 yang kami uji menggunakan algoritma genetik yang kami buat:



Gambar Test Case 2 beserta Encodingnya

Lalu hasil yang didapatkan oleh program ini adalah sebagai berikut:



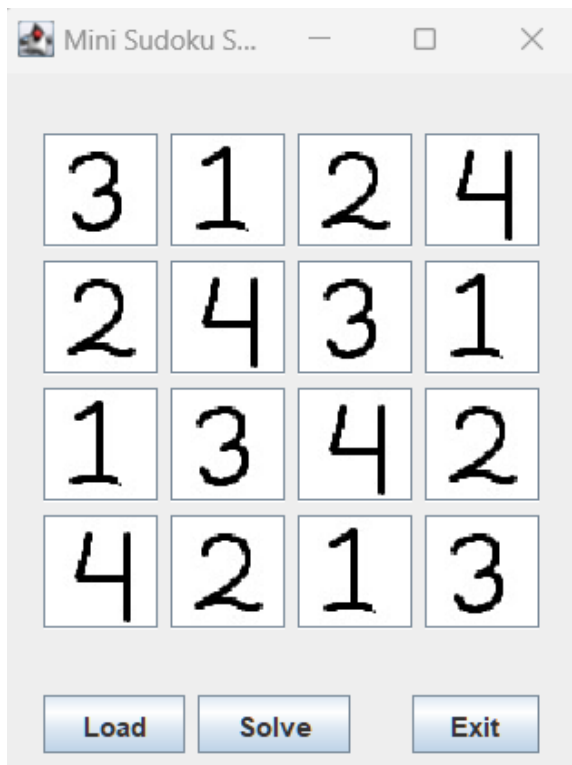
Gambar Test Case 2 beserta Encodingnya

Hasil tersebut didapatkan dengan melakukan percobaan eksperimen dengan parameter generasi maksimum adalah 40. Berikut hasil keluaran yang ditampilkan oleh program:

```
The list is: [2, 1, 2, 1, 4, 3, 3, 4, 3, 4, 4]
The fitness is: 7
-----
| 3 | 1 | 2 | 1 |
-----
| 2 | 1 | 4 | 3 |
-----
| 3 | 4 | 3 | 2 |
-----
| 4 | 2 | 1 | 4 |
-----
```

Dari keluaran yang ditampilkan oleh program untuk test case yang kedua, Program menemukan fitness terbaik pada generasi ke 26 dengan nilai fitness 7. Namun hasil dari pengujian tersebut bukanlah hasil yang diharapkan

Contoh hasil yang diharapkan:



Gambar dari test case 2 dengan hasil yang diharapkan

5.3 Test case 23

Berikut test case 23 yang kami uji menggunakan algoritma genetik yang kami buat:

The image shows a screenshot of a software application titled "Mini Sudoku S...". The application features a 4x4 grid for a Mini Sudoku puzzle. The grid contains the following numbers:

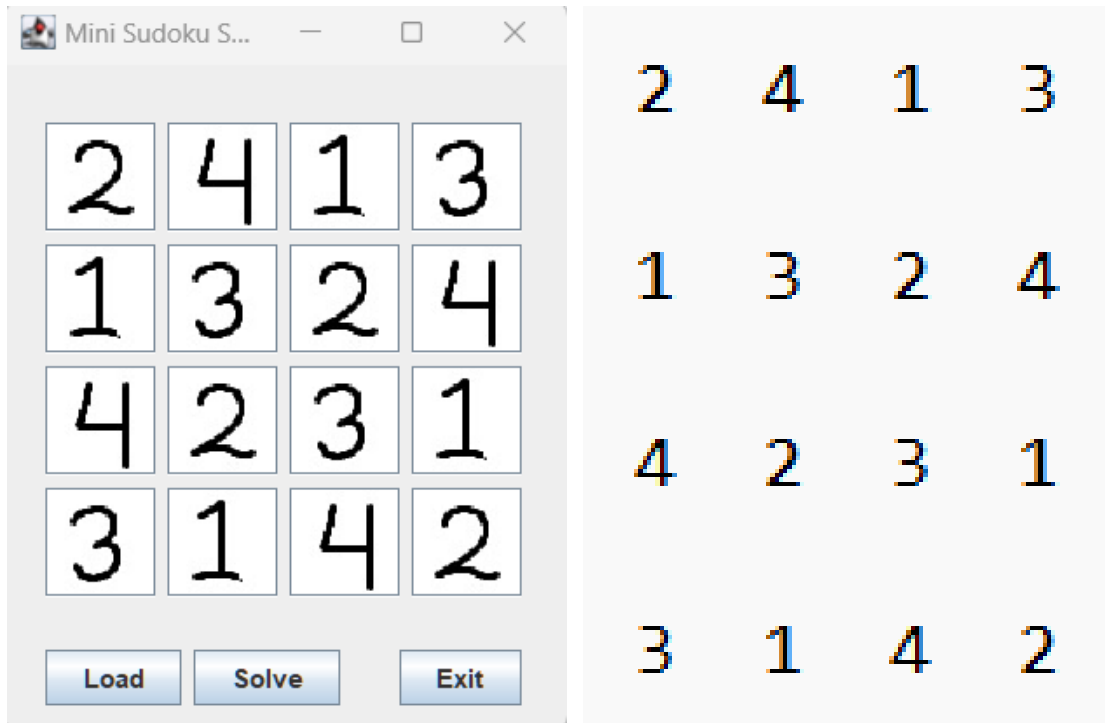
2		1	
1	3		
4			1
3		4	

Below the grid are three buttons: "Load", "Solve", and "Exit". To the right of the application window, the encoding of the grid is displayed as a 4x4 matrix of numbers and signs:

2	-1	1	-1
1	3	-1	-1
4	-1	-1	1
3	-1	4	-1

Gambar Test Case 23 beserta Encodingnya

Lalu hasil yang didapatkan oleh program ini adalah sebagai berikut:



Gambar Hasil Test Case 23 beserta Encodingnya

Hasil tersebut didapatkan dengan melakukan percobaan eksperimen dengan parameter generasi maksimum adalah 40. Berikut hasil keluaran yang ditampilkan oleh program:

```

src > minisudokuga > J Chromosome.java M Chromosome > toBoard(int)
subBlockDuplications++;
}
}
return subBlockDuplications;
}
int fitness(){
return rowCheck()+colCheck()+subCheck();
}

```

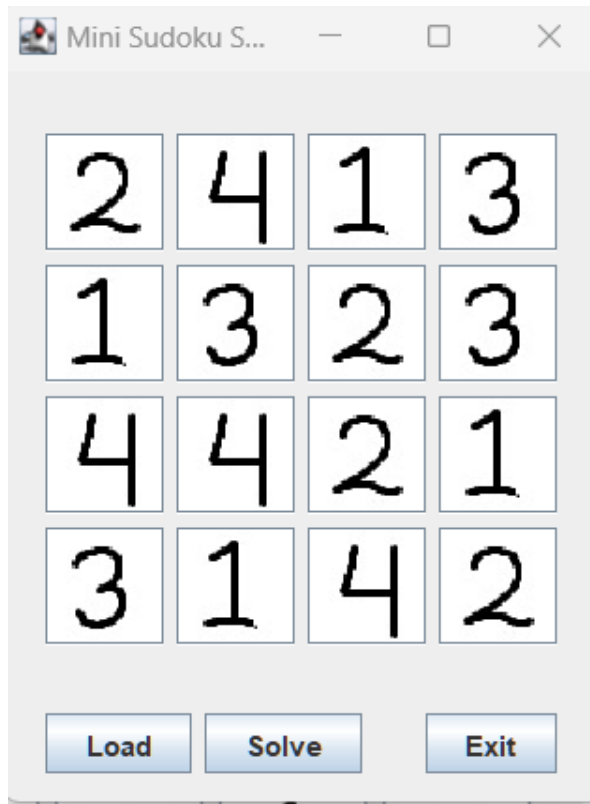
```

| 3 | 2 | 4 | 4 |
-----
The list is: [4, 3, 2, 4, 2, 3, 1, 2]
The fitness is: 0
| 2 | 4 | 1 | 3 |
| 1 | 3 | 2 | 4 |
| 4 | 2 | 3 | 1 |
| 3 | 1 | 4 | 2 |

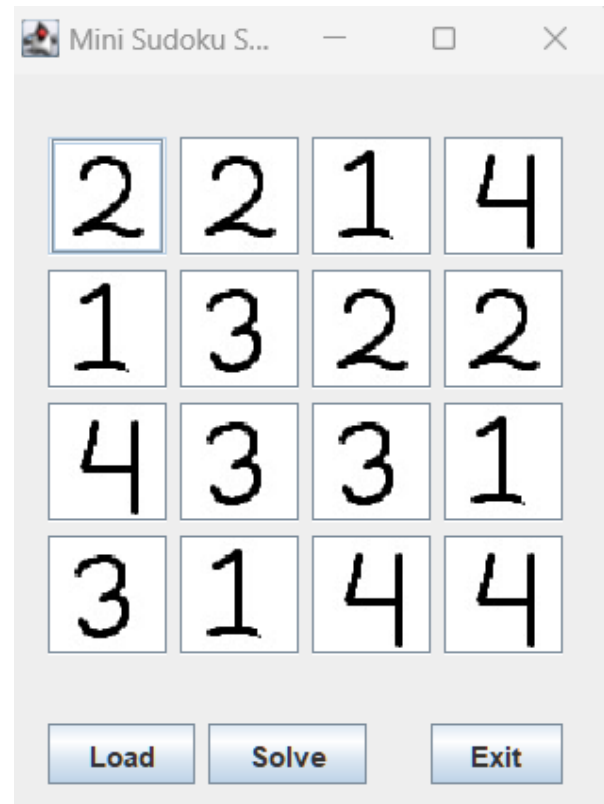
```

Dari keluaran yang ditampilkan oleh program untuk test case yang ke-23, Program menemukan fitness terbaik pada generasi ke 23 dengan nilai fitness 0. Dari hasil pengujian tersebut, hasil akhir sudah sesuai dengan yang diharapkan.

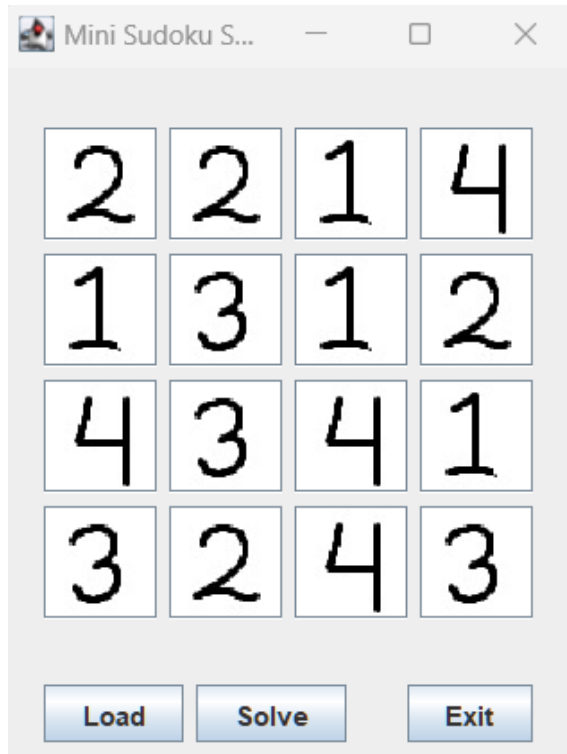
Untuk gambar hasil percobaan ke-17 hingga 22 akan ditampilkan berikut ini:



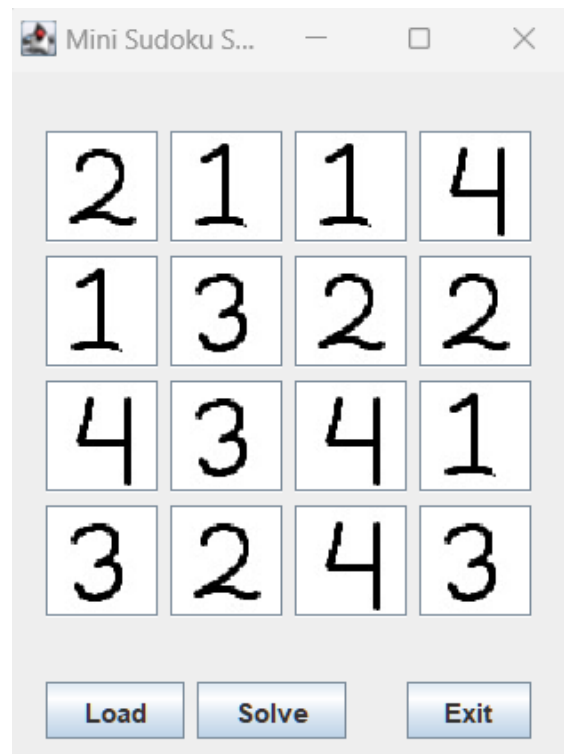
Percobaan 17: Fitness: 8



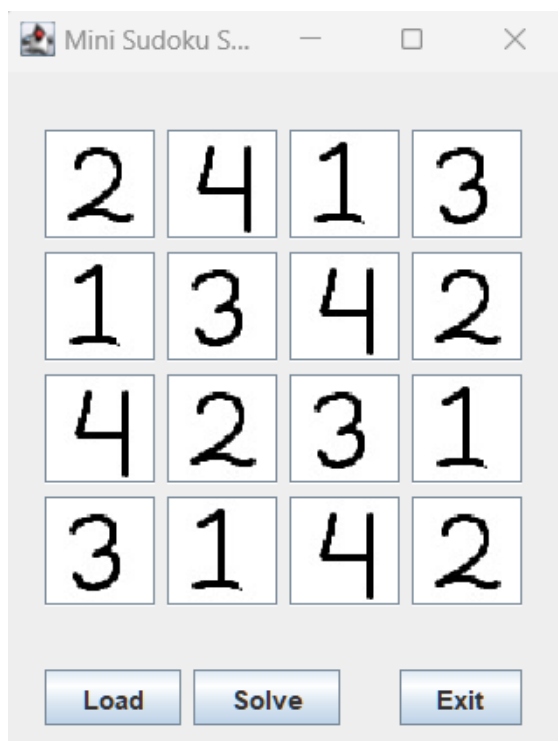
Percobaan 18: Fitness: 10



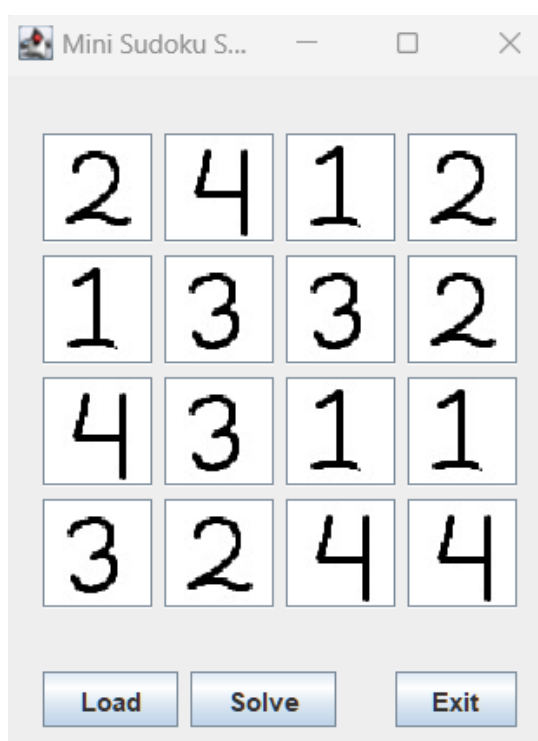
Percobaan 19: Fitness: 12



Percobaan 20: Fitness: 10



Percobaan 21: Fitness: 2



Percobaan 22: Fitness: 11

5.4 Test Case lainnya

Untuk hasil test case yang lain, hanya akan ditampilkan hasil fitness nya saja, dan tidak beserta hasil screenshotnya:

- Test Case 3: Fitness: 7; Generasi ke: 30
- Test Case 4: Fitness: 9; Generasi ke: 11
- Test Case 5: Fitness: 4; Generasi ke: 19
- Test Case 6: Fitness: 4; Generasi ke: 13
- Test Case 7: Fitness: 8; Generasi ke: 5
- Test Case 8: Fitness: 6; Generasi ke: 10
- Test Case 9: Fitness: 9; Generasi ke: 39
- Test Case 10: Fitness: 8; Generasi ke: 27

Dan untuk sisa test case yang lain ada yang bisa mendapatkan nilai fitness 0, dan juga ada yang tidak berhasil.

BAB VI

KESIMPULAN

6.1. Kesimpulan:

Sudoku berukuran 4x4 menawarkan tantangan yang menarik dalam penggunaan algoritma genetik untuk diselesaikan. Pada proyek yang kami jalani, kami menemukan bahwa kesuksesan dalam menyelesaikan Sudoku 4x4 dengan algoritma genetik sangat bergantung pada perancangan fungsi kecocokan (fitness function) yang berkualitas. Sayangnya, dalam pengerjaan tugas kali ini, kami menghadapi kendala karena terbatasnya kualitas fungsi kecocokan yang kami implementasikan. Hal ini menyebabkan algoritma kami menjadi sangat tergantung pada elemen-elemen acak yang ditempatkan oleh program.

Efek dari kurangnya kualitas fungsi kecocokan ini sangat nyata dalam hasil pengujian kami. Dari tiga test case yang kami lampirkan beserta test case lainnya, hanya satu di antaranya yang berhasil mencapai tingkat kecocokan yang baik dan mampu menyelesaikan permainan Sudoku 4x4 dengan sukses dalam 40x percobaan. Sisanya, dengan fungsi kecocokan yang kurang optimal, menghasilkan hasil yang tidak sesuai dengan harapan kami. Kendala ini menjadi bagian integral dari pengembangan algoritma genetik, yang mempertegas pentingnya perancangan yang matang dan penyesuaian yang cermat terhadap elemen-elemen kunci seperti fungsi kecocokan, sehingga mencapai hasil yang lebih konsisten dan memuaskan dalam menyelesaikan permasalahan Sudoku 4x4 maupun permainan lainnya.

6.2. Saran:

Berdasarkan hasil pengamatan yang telah kami lakukan, kami ingin memberikan saran bahwa dalam menyelesaikan Sudoku berukuran 4x4 atau bahkan dalam pengembangan game apapun dengan menggunakan algoritma genetik, sangatlah penting untuk mengambil pertimbangan yang matang terhadap beberapa elemen kunci, yaitu fungsi fitness, proses crossover, metode mutasi, dan pemilihan parent. Dengan memperhatikan faktor-faktor ini secara cermat, kami percaya bahwa Pemain dapat meningkatkan efisiensi dan efektivitas algoritma genetik dalam menyelesaikan permasalahan yang kompleks seperti Sudoku atau permainan lainnya.

6.3. Referensi

<https://en.wikipedia.org/wiki/Sudoku>

<https://www.createclassicsudoku.com/MakeYourOwnSudoku4by4.jsp>

<https://www.sudokuonline.io/kids/numbers-4-4>

https://en.wikipedia.org/wiki/Genetic_algorithm