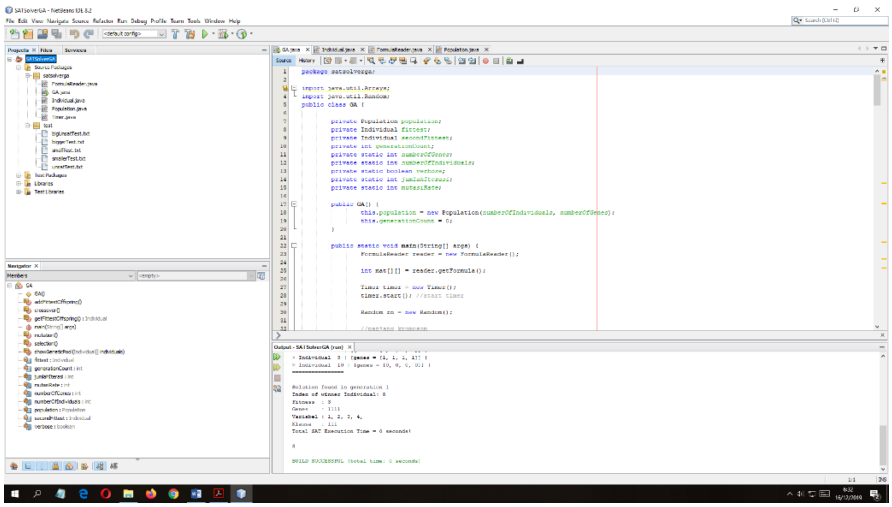
BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

Implementasi adalah tahap selanjutnya untuk mengerjakan apa yang telah dirancang pada tahapan di bab sebelumnya. Implementasi akan menunjukkan apakah perancangan yang telah dilakukan sebelumnya dapat berjalan dan dapat digunakan dengan baik. Pada tahapan ini akan dibuat SAT *Solver* yang merepresentasikan implementasi penyelesaian pendekatan solusi SAT *Problem* dengan menggunakan Algoritma Genetika menggunakan bahasa pemrograman JAVA.

4.1.1 Tools/Netbeans IDE

Netbeans merupakan proyek yang disponsori oleh Sun Microsystem sejak tahun 2000. Pada proyek tersebut dihasilkan dua produk, yaitu Netbeans IDE dan Netbeans Platform. Netbeans IDE merupakan produk yang digunakan untuk melakukan pemrograman, baik menulis kode, mengompilasi, mencari kesalahan, dan mendistribusikan program. Sedangkan Netbeans Platform adalah sebuah modul yang merupakan kerangka awal dalam membangun aplikasi desktop yang besar. Netbeans merupakan salah satu IDE yang dapat digunakan dalam melakukan pemrograman Java. Selain itu, Netbeans menyediakan paket yang lengkap dalam pemrograman, dari pemrograman standar, enterprise, dan pemrograman perangkat mobile yang dapat berjalan di berbagai macam platform. Tampilan Netbeans IDE dapat dilihat pada Gambar 4.1.



Gambar 4.1 Tampilan Netbeans IDE

4.1.2 *Method* read

Method Formula read berfungsi untuk membaca formula logika proposisi dalam bentuk CNF dan format .txt. Formula tersebut disimpan pada pada tipe data String bernama file. Nantinya sebelum algoritma genetika berjalan *Method* main pada Kelas GA akan memberikan nama file CNF ke variabel file. Selain itu *Method* read juga memberi nilai kepada variabel numVars, numClauses, dan formula. Adapun implementasi program JAVA untuk *Method* read dapat dilihat pada Gambar 4.2.

```
public void read(String file) {

        try {
            Scanner fileScanner = new Scanner(new File(file));
            Pattern pattern = Pattern.compile("p cnf");
            while (fileScanner.findInLine(pattern) == null)
                 fileScanner.nextLine();
                 numVars = fileScanner.nextInt();
                 numClauses= fileScanner.nextInt();
                 formula = new int[numClauses][];
                 LinkedList<Integer> curLine = new LinkedList<>();
            for (int curRow = 0, curNum; fileScanner.hasNextInt();) {
                 curNum = fileScanner.nextInt();
            if (curNum == 0) {
               formula[curRow] = curLine.stream().mapToInt(i->i).toArray();
               curRow++;
               curLine.clear();
               continue;
            }
                 curLine.add(curNum);
            }
            fileScanner.close();

        } catch (FileNotFoundException ex) {
            System.out.println("ERROR: Unable to read " + file + ".");
            System.exit(0);
        }
}
```

Gambar 4.2 *Method* read

4.1.3 *Method* main (Initialization)

Method main pada bagian *Initialization* berfungsi untuk merancang struktur sebuah populasi. Perancangan tersebut dilakukan dengan cara pertama membuat object Kelas GA bernama demo, kemudian membuat object kelas Populasi dan memberikan parameter kepada object tersebut. Parameter tersebut adalah numberOfIndividuals yang memberikan nilai panjang gen untuk individu/kromosom dan numberOfGenes yang memberikan banyaknya

individu dalam populasi tersebut kedua parameter tersebut memiliki tipe data integer. Adapun implementasi program JAVA untuk *Method* main pada bagian *Initialization* dapat dilihat pada Gambar 4.3.

```
  public static void main(String[] args) {

       GA demo = new GA();

       demo.populasi = new Populasi(numberOfIndividuals, numberOfGenes);

}
```

Gambar 4.3 *Method* main (Initialization)

Method Populasi berfungsi untuk membuat sebuah populasi yang sesuai dengan parameter yang diberikan dari *Method* main di Kelas GA. Parameter tersebut meliputi panjang gen untuk individu/kromosom yang disimpan pada variabel geneLength dengan tipe data integer dan banyaknya individu dalam populasi tersebut yang disimpan pada variabel popSize tipe data integer. Selain itu *Method* Populasi membuat array individu baru dengan panjang sesuai geneLength. Adapun implementasi program JAVA untuk *Method* Populasi dapat dilihat pada Gambar 4.4.

```
   public Populasi(int popSize, int geneLength) {
       super();
       this.popSize = popSize;
       this.geneLength = geneLength;
       this.individuals = new Individual[popSize];

       for (int i = 0; i < popSize; i++) {
           individuals[i] = new Individual(geneLength);
       }
   }
```

Gambar 4.4 *Method* Populasi

Method Individual berfungsi untuk membentuk array integer genes sebanyak geneLength. Lalu memberikan nilai pada genes. Nilai yang diberikan adalah bilangan integer 0 sampai 1 untuk merepresentasikan solusi dari logika proposisi. Pemberian nilai dilakukan secara acak menggunakan metode acak pada JAVA. Adapun implementasi program JAVA untuk *Method* Populasi dapat dilihat pada Gambar 4.5.

```
   public Individual(int geneLength) {

       this.geneLength = geneLength;
       this.genes = new int[geneLength];

       for (int i = 0; i < genes.length; i++) {
          genes[i] = ThreadLocalRandom.current().nextInt(2);
       }

       fitness = 0;
   }
```

Gambar 4.5 *Method* Individual

4.1.4 *Method* calcFitness (Evaluation)

Method Evaluation berfungsi untuk mengevaluasi/menghitung nilai *fitness* individu. Penghitungan nilai *fitness* tersebut dilakukan dengan cara membuat object reader. Kemudian mengisikan variabel mat yang merupakan data 2D arrays dengan variabel formula pada *Method* read. Kemudian mengisikan nilai variabel mat dengan variabel genes. Setelah terbentuk variabel mat dengan nilai yang baru dilakukan penjumlahan tiap baris yang menghasilkan keluaran bilangan integer 0 atau 1 dan kemudian disimpan di variabel zum. Variabel zum nanti akan dijumlahkan seluruh nilainya dan menjadi nilai fitness individu tersebut. Adapun implementasi program JAVA untuk *Method* Evaluation dapat dilihat pada Gambar 4.6.

```
public void calcFitness(){
            PembacaFormula reader = new PembacaFormula();
            int mat[][] = reader.getFormula();

            for(int i = 0; i < genes.length; i++){
                for (int j = 0; j < mat.length; j++){
                    for (int k = 0; k < mat[j].length; k++){
                        if(Math.abs(mat[j][k])== i+1){
                            if(mat[j][k] < 0){
                                if(genes[i] == 0){
                                    mat[j][k] = genes[i]+1;
                                }else{
                                    mat[j][k] = genes[i]-1;
                                }
                            }else{
                                mat[j][k] = genes[i];
             }   }   }   }  }


            int zum = 0;
            int arr[] = new int[1000];
            for (int a = 0; a < mat.length; a++){
                for (int b = 0; b < mat[a].length; b++){
                    zum += mat[a][b];
                }
                    if (zum == 0 ){
                        zum = 0;
```

```
                        arr[a] = zum;
                    }else{
                        zum = 1;
                        arr[a] = zum;
                    }

                zum = 0;
            }

            int sum = IntStream.of(arr).sum();
            fitness = sum;
    }
```

Gambar 4.6 *Method* calcFitness

4.1.5 *Method* selection

Method Selection berfungsi untuk memilih dua individu dengan nilai *fitness* terburuk, lalu menggantinya dengan dua individu dengan nilai *fitness* terbaik. Adapun implementasi program JAVA untuk *Method* Selection dapat dilihat pada Gambar 4.7.

```
void selection() {

fittest.calcFitness();
secondFittest.calcFitness();

int leastFittestIndex = populasi.getLeastFittestIndex();
int secondLeastFittestindex = populasi.getsecondLeastFittestIndex();

populasi.getIndividuals()[leastFittestIndex]=        getFittestOffspring();
populasi.getIndividuals()[secondLeastFittestIndex]=
getSecondFittestOffspring();

}
```

Gambar 4.7 *Method* selection

Method getLeastFittestIndex dan getSecondLeastFittestIndex adalah *Method* yang digunakan untuk memilih dua individu dengan nilai *fitness* terburuk. Adapun implementasi program JAVA untuk *Method* getLeastFittestIndex dan getSecondLeastFittestIndex dapat dilihat pada Gambar 4.8.

```
public int getLeastFittestIndex() {
       int minFitVal = Integer.MAX_VALUE;
       int minFitIndex = 0;
       for (int i = 0; i < individuals.length; i++) {
           if (minFitVal >= individuals[i].getFitness()) {
              minFitVal = individuals[i].getFitness();
              minFitIndex = i;
           }
       }
       return minFitIndex;
```

```
    }

public int getSecondLeastFittestIndex() {
       int minFitVal = Integer.MAX_VALUE;
       int minFitIndex = 0;
       int secondMinFitVal = Integer.MAX_VALUE;
       int secondMinFitIndex = 0;

       for (int i = 0; i < individuals.length; i++) {
          if (minFitVal >= individuals[i].getFitness()) {
             minFitVal = individuals[i].getFitness();
             minFitIndex = i;
          }
       }
       for (int i = 0; i < individuals.length; i++) {
          if (secondMinFitVal >= individuals[i].getFitness()) {
             if(secondMinFitVal >= minFitVal){
                secondMinFitVal = individuals[i].getFitness();
                secondMinFitIndex = i;
             }
          }
    }
       return secondMinFitIndex;
}
```

Gambar 4.8 *Method* getLeastFittestIndex dan getSecond LeastFittestIndex

Sedangkan *Method* selectFittest dan selectSecondFittest digunakan untuk memilih individu dengan nilai *fitness* terbaik. Adapun *Method* selectFittest dan selectSecondFittest dapat dilihat pada Gambar 4.9.

```
 public Individual selectFittest() {
    int maxFit = Integer.MIN_VALUE;
    int maxFitIndex = 0;
    for (int i = 0; i < individuals.length; i++) {
       if (maxFit <= individuals[i].getFitness()) {
          maxFit = individuals[i].getFitness();
          maxFitIndex = i;
       }
 }
    fittestScore = individuals[maxFitIndex].getFitness();

    try {
       return (Individual) individuals[maxFitIndex].clone();
    } catch (CloneNotSupportedException e) {
       e.printStackTrace();
    }
       return null;
    }

 public Individual selectSecondFittest() {
    int maxFit1 = 0;
    int maxFit2 = 0;
    for (int i = 0; i < individuals.length; i++) {
    if (individuals[i].getFitness() > individuals[maxFit1].getFitness()) {
```

```
            maxFit2 = maxFit1;
            maxFit1 = i;
}else if (individuals[i].getFitness()>individuals[maxFit2].getFitness()) {
            maxFit2 = i;
         }
      }
      try {
         return (Individual) individuals[maxFit2].clone();
      } catch (CloneNotSupportedException e) {
         e.printStackTrace();
      }
      return null;
}
```

Gambar 4.9 *Method* selectFittest dan selectSecondFittest

4.1.6 *Method* crossover

Method crossover berfungsi untuk menyilangkan/mengawinkan dua individu dengan nilai *fitness* terbaik. Pertama dibangkitkan angka random untuk memberikan nilai pada variabel crossOverPoint yang merupakan representasi cut-point untuk kedua individu tersebut. Selanjutnya akan terjadi pertukaran nilai antar individu yang akan menghasilkan dua individu baru (*offspring*). Adapun implementasi program JAVA untuk *Method* crossover dapat dilihat pada Gambar 4.10.

```
   void crossover() {

      Random rn = new Random();


      int crossOverPoint = rn.nextInt(this.numberOfGenes);

      for (int i = 0; i < crossOverPoint; i++) {
         int temp = fittest.getGenes()[i];
         fittest.getGenes()[i] = secondFittest.getGenes()[i];
         secondFittest.getGenes()[i] = temp;

      }

   }
```

Gambar 4.10 *Method* crossover

Method addBestOffspring berfungsi untuk memilih individu dengan nilai *fitness* terburuk dan menggantinya dengan individu baru hasil persilangan (*offspring*). Adapun implementasi program JAVA untuk *Method* addBestOffspring dapat dilihat pada Gambar 4.11.

```
void addBestOffspring() {

    fittest.calcFitness();
    secondFittest.calcFitness();
    int leastFittestIndex = populasi.getLeastFittestIndex();
    populasi.getIndividuals()[leastFittestIndex]=getFittestOffspring();
}
```

Gambar 4.11 *Method* addBestOffspring

4.1.7 *Method* mutation

Method mutation berfungsi untuk membuat mutasi/kelainan gen. Pada *Method* ini akan dipilih dua individu terbaik kemudian ditentukan *point mutation* menggunakan metode acak JAVA. Pada *Method* ini dua individu tersebut akan mengalami perubahan nilai pada point yang telah ditentukan. Adapun implementasi program JAVA untuk *Method* mutation dapat dilihat pada Gambar 4.12.

```
 void mutation() {

       Random rn = new Random();


       int mutationPoint = rn.nextInt(this.numberOfGenes);

       if (fittest.getGenes()[mutationPoint] == 0) {
           fittest.getGenes()[mutationPoint] = 1;
       } else {
           fittest.getGenes()[mutationPoint] = 0;
       }

       mutationPoint = rn.nextInt(this.numberOfGenes);

       if (secondFittest.getGenes()[mutationPoint] == 0) {
           secondFittest.getGenes()[mutationPoint] = 1;
       } else {
           secondFittest.getGenes()[mutationPoint] = 0;
       }
}
```

Gambar 4.12 *Method* mutation

4.1.8 *Method* main (Perulangan)

Method main pada bagian perulangan berfungsi untuk mengeksekusi perulangan algoritma genetika dengan konsep pendekatan solusi. Perulangan dilakukan sampai salah satu kondisi berhenti terpenuhi yaitu maksimal iterasi terpenuhi atau solusi eksak sudah ditemukan. Dalam perulangan akan dilakukan pemanggilan *Method* selection, crossover, mutation, dan calculateFitness. Adapun implementasi program JAVA untuk *Method* main pada bagian perulangan dapat dilihat pada Gambar 4.13.

```
public static void main(String[] args) {

while(demo.generationCount<maxIteration&&demo.populasi.getFittestScore() <
reader.getClauses()) {

          ++demo.generationCount;


          demo.selection();


          demo.crossover();


          if (rn.nextInt(99) <= mutationRate-1) {
             demo.mutation();
          }

          demo.addBestOffspring();

          demo.populasi.calculateFitness();


       }

}
```

Gambar 4.13 *Method* main (perulangan)

4.2 Pengujian

Pada tahapan ini SAT *Solver* yang telah dikembangkan akan diuji sehingga dapat diketahui hasilnya sudah sesuai dengan kebutuhan atau belum. Dalam penelitian ini, pengujian terhadap SAT S*olver* yang telah dibentuk adalah menyelesaikan beberapa SAT *Problem*. Setiap SAT *Problem* yang akan diujikan disimpan dalam sebuah *file* teks (CNF *file*). Format penulisan teks tersebut mengikuti aturan penulisan seperti di kompetisi SAT (http://satcompetition.org).

Dalam penelitian ini, pengujian dilakukan dengan menggunakan CNF *file* yang digunakan untuk menguji SAT Solver dengan *resource* relatif kecil. CNF *file* tersebut diambil dari web (https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html). CNF *file* yang digunakan dalam pengujian dapat dilihat pada Tabel 4.1.

Tabel 4.1 CNF Files yang digunakan untuk pengujian

No	Nama	Variabel	Klausa	Nilai Kebenaran Formula
1	simple_v3_c2.cnf	3	2	*satisfiable*
2	quinn.cnf	16	18	*satisfiable*
3	hole6.cnf	42	133	*unsatisfiable*
4	aim-50-1_6-yes1-4.cnf	50	80	*satisfiable*
5	dubois20.cnf	60	160	*unsatisfiable*
6	dubois21.cnf,	63	168	*unsatisfiable*
7	par8-1-c.cnf	64	254	*satisfiable*
8	dubois22.cnf,	66	176	*unsatisfiable*
9	aim-100-1_6-no-1.cnf	100	160	*unsatisfiable*
10	zebra_v155_c1135.cnf	155	1135	*satisfiable*

Pengujian dilakukan dengan menggunakan *hardware*/perangkat keras berupa laptop dengan prosesor i5-7200U @2,5Ghz dan RAM 8 GB dan parameter penelitian sesuai dengan perancangan pada BAB III, untuk lebih detail adalah sebagai berikut:

1. Rasio mutasi sebesar 25%.
2. Rasio *crossover* sebsar 100 %.
3. Batasan iterasi sebanyak 1000 kali/generasi.
4. Banyak individu adalah 10.
5. Setiap CNF *file* akan dilakukan pengujian sebanyak 10 kali, yang merupakan standar untuk pengujian skala kecil (Bouhmala, 2018).

4.3 Pembahasan

SAT *Solver* ini berhasil menyelesaikan keseluruhan pengujian CNF *file* pertama dan CNF *file* kedua dalam waktu mendekati 0 detik. Dalam keseluruhan pengujian kedua CNF *file* tersebut berhasil memberikan solusi eksak/*satisfiable* sedangkan CNF *file* yang lain diselesaikan menggunakan solusi pendekatan.

Secara umum, kinerja SAT *Solver* ini sudah cukup baik karena dapat menyelesaikan seluruh seluruh CNF *file* yang diberikan dengan akurasi di atas 90%. Waktu yang diperlukan SAT *Solver* untuk menyelesaikan CNF *file* relatif singkat yaitu masih dalam kurun waktu detik. Namun perlu digaris bawahi bahwa kasus yang diselesaikan masih tergolong kecil. SAT *Solver* ini perlu diuji dengan SAT *Problem* yang lebih besar untuk melihat potensi sesungguhnya.

Selain itu rata-rata waktu penyelesaian CNF *file* beragam tapi masih mengikuti pola semakin besar klausa atau literal maka semakin lama juga waktu pengerjaaannya. Penentuan nilai kebenaran terhadap formula logika proposisi juga sudah mengikuti pola rancangan yang sudah dijelaskan pada BAB III. Rata-rata hasil 10 kali pengujian dapat dilihat pada Tabel 4.2. Keseluruhan pengujian dapat dilihat pada bagian Lampiran.

Tabel 4.2 Hasil Pengujian

No	Nama	Akurasi	Generasi	Waktu	Nilai Kebenaran Formula
1	simple_v3_c2.cnf	100%	1	0.003 detik	*Satisfiable*
2	quinn.cnf	100%	36.9	0.129 detik	*Satisfiable*
3	hole6.cnf	97.067%	1000	7 detik	Pendekatan
4	aim-50-1_6-yes1-4.cnf	94.25%	1000	6 detik	Pendekatan
5	dubois20.cnf	94%	1000	8 detik	Pendekatan
6	dubois21.cnf	92.737%	1000	7 detik	Pendekatan
7	par8-1-c.cnf	93.503%	1000	11 detik	Pendekatan
8	dubois22.cnf	93.636%	1000	9 detik	Pendekatan
9	aim-100-1_6-no-1.cnf	93.362%	1000	11 detik	Pendekatan
10	zebra_v155_c1135.cnf	93.391%	1000	42 detik	Pendekatan

4.4 Kelebihan dan Kekurangan Penelitian

Dalam suatu penelitian pasti akan memiliki kelebihan dan kekurangan termasuk penelitian ini. Penelitian ini mempunyai kelebihan dan kekurangan baik pada tahapan identifikasi masalah, analisis kebutuhan, perancangan, implementasi, ataupun pengujian penelitian.

4.4.1 Kelebihan Penelitian

Penelitian ini memiliki beberapa kelebihan yang cukup baik. Kelebihan tersebut diantaranya adalah:

1. Perancangan yang dilakukan berhasil menggabungkan Algoritma Genetika untuk menyelesaiakan SAT *Problem* dengan solusi pendekatan.
2. Berhasil diimplementasikan SAT *Solver* dengan menggunakan bahasa pemrograman JAVA.
3. Dari pengujian dapat disimpulkan bahwa SAT *Solver* yang dibentuk sudah memenuhi analisis kebutuhan dan perancangan.

4.4.2 Kekurangan Penelitian

Penelitian ini memiliki beberapa kelemahan dikarenakan adanya suatu keterbatasan. Kelemahan tersebut diantaranya adalah:

1. Identifikasi masalah sebagian besar hanya dilakukan dengan membaca jurnal-jurnal penelitian terdahulu
2. Pada implementasi SAT *Solver* tidak menggunakan *user interface* yang menarik.
3. Pengujian belum dapat dilakukan secara optimal karena CNF *file* yang digunakan pada SAT *Competition* dan menjadi standar pengujian SAT internasional memiliki kompleksitas yang tinggi. Permintaan minimum *hardware*/perangkat keras yang diminta sangatlah tinggi yaitu prosesor Intel(R) Xeon(R) CPU E5-2609 dan memori internal sebesar 120GB. Sehingga dipilihlah CNF *file* yang digunakan untuk menguji SAT *Solver* dengan *resource* kecil.