Solusi pendekatan SAT Problem dengan Algoritma Genetika

Muh Arnesz Setiawan
Jurusan Informatika, Fakultas Teknologi Industri
Universitas Islam Indonesia
Yogyakarta
16523202@students.uii.ac.id

Taufiq Hidayat
Jurusan Informatika, Fakultas Teknologi Industri
Universitas Islam Indonesia
Yogyakarta
taufiq.hidayat@uii.ac.id

***Abstrak*—SAT Problem yang merupakan permasalahan NP-*Complete* yang sulit dikerjakan dengan cara konvensional dan telah memiliki banyak aplikasi (SAT Solver) untuk menyelesaikannya. Namun dari seluruh SAT Solver yang ada belum ditemukan yang menawarkan solusi yang bukan eksak (pendekatan). Penelitian ini bertujuan untuk mengembangkan SAT Solver menggunakkan pemrograman JAVA dan konsep pendekatan solusi menggunakan Algoritma Genetika. Setelah dilakukan pengujian, kesimpulan yang dapat diambil adalah SAT Solver dari penelitian ini dapat menyelesaikan SAT Problem. Salah satu problem yang diujikan adalah sebuah CNF *file* dengan 16 variabel dan 18 klausa.**

Kata kunci—SAT Problem, SAT Solver, Algoritma Genetika.

I. PENDAHULUAN

Dalam kehidupan sehari-hari terdapat berbagai macam permasalahan yang terjadi. Beberapa permasalahan tersebut dapat ditangani secara matematis, salah satu metodenya adalah SAT Problem (*Boolean Satisfiability Problem*). SAT Problem dapat digunakan untuk menentukan permasalahan kombinasi nilai kebenaran semua variabel proposisi dalam sebuah klausa/formula logika proposisional, sehingga membuat klausa/formula logika proposisional tersebut bernilai benar [1].

Untuk mempermudah proses penyelesaian masalah menggunakan metode SAT Problem perlu adanya SAT Solver. SAT Solver adalah perangkat lunak atau *software* yang digunakan untuk menyelesaikan SAT Problem, yaitu untuk menentukan apakah sebuah formula logika proposisi itu *satisfiable* atau *unsatisfiable*. Formula logika proposisi yang diselesaikan dengan SAT Solver harus dalam bentuk CNF [2].

Dalam berbagai kasus dan permasalahan yang kompleks pada dunia nyata, SAT Solver terkadang akan mengalami kesulitan. Hal ini disebabkan karena pada awalnya SAT Solver dibuat dengan menggunakan algoritma basic dan hanya untuk mengatasi permasalahan yang *simple* (mudah). Oleh karena itu optimasi algoritma pembentuk SAT Solver akan menjadi *essential* (penting). Peningkatan SAT Solver berarti kualitas solusi yang didapatkan akan lebih baik, proses komputasi yang memakan waktu lebih sedikit dan biaya (*resource*) yang diperlukan akan jadi jauh lebih rendah daripada SAT Solver sebelumnya.

Selama beberapa dekade terakhir penelitian tentang algoritma pembentuk SAT Solver yang efisien dan *scalable* untuk menyelesaikan SAT Problem berkembang dengan pesat. Perkembangan ini telah banyak berkontribusi pada kemajuan teknologi terutama dalam hal perkembangan kemampuan kita (manusia) untuk secara otomatis memecahkan masalah yang melibatkan puluhan ribu variabel dan jutaan klausa (*constraint*) [3]. Dalam dunia nyata salah satu contoh penggunaan SAT Solver adalah dalam permasalahan EDA (*Electronic Design Automation*). Permasalahan tersebut meliputi *formal equivalence checking, model checking, formal verification of pipelined microprocessors*[4], *automatic test pattern generation, routing of FPGAs*[5], perencanaan dan permasalahan penjadwalan, dan sebagainya. Oleh karena itu SAT Solver sekarang dianggap sebagai komponen penting dalam desain EDA.

Belakangan ini tren perkembangan algoritma pembentuk SAT Solver tersebut mengarah ke *Soft Computing*. Salah satu cabang dari *Soft Computing* adalah Algoritma Genetika/Genetic Algorithm (GA). Algoritma genetika adalah algoritma metaheuristik (pencarian) yang terinspirasi teori Darwin mengenai proses seleksi alam dan termasuk ke dalam kelas yang lebih besar yaitu *evolutionary algorithm* (EA). Algoritma Genetika biasanya digunakan untuk menghasilkan solusi berkualitas tinggi untuk masalah optimasi dan pencarian dengan mengandalkan operator yang terinspirasi alam seperti mutasi, *crossover* dan seleksi [6]. Karena dapat menghasilkan solusi berkualitas tinggi dengan *resource* relatif rendah Algoritma Genetika menjadi kandidat yang kuat untuk menjadi algoritma pembentuk SAT Solver.

Sebenarnya sudah ada banyak SAT Solver yang dikembangkan menggunakan algoritma yang berbeda-beda seperti: GRASP[9], Satz[8], WalkSat[7], zChaff dan mChaff[10], bahkan sebenarnya sudah ada penelitian mengenai SAT Solver menggunakan Algoritma Genetika[12]. Namun semua SAT Solver tersebut masih membutuhkan *resource* yang besar untuk dijalankan serta memakan banyak waktu dalam proses eksekusinya karena solusi yang ditawarkan berupa eksak. Eksak yang dimaksud adalah dalam penyelesaian SAT Problem dalam bentuk CNF harus menempatkan nilai variabel proposisi sehingga semua klausa harus bernilai benar.

Sedangkan pendekatan yang dimaksud adalah memaksimalkan klausa yang bernilai benar dalam batas atas iterasi ditetapkan. solusi terbaik pada akhir iterasi menjadi solusi permasalahan. Hal ini akan sangat menguntungkan karena pendekatan merupakan masalah optimasi dan Algoritma Genetika adalah algoritma optimasi artinya *resource* yang diperlukan hanya akan bergantung seberapa lama proses optimasi berlangsung.

Oleh karena itu penelitian ini akan mencoba mengembangkan aplikasi (SAT Solver) dengan menggunakan pemrograman JAVA dan konsep pendekatan solusi menggunakan Algoritma Genetika. Penerapan konsep pendekatan solusi diharapkan dapat meningkatkan kinerja SAT Solver dengan Algoritma genetika.

II. LANDASAN TEORI

A. Tinjauan Pustaka

Dasar teori dalam penyelesaian *SAT Problem* ini adalah teori yang menjadi acuan dalam penelitian yang dilakukan. Seperti yang disebutkan dalam penjelasan di bawah ini:

1. Penelitian yang dilakukan E. Marchiori and C. Rossi dengan judul "*A Flipping Genetic Algorithm for Hard 3-SAT Problems*"[12]. Penelitian ini dilakukan untuk mengembangkan SAT Solver dengan Algoritma Genetika namun solusi/keluaran yang dihasilkan adalah eksak, bukan pendekatan. Algoritma yang digunakan dalam penelitian ini bukanlah murni Algoritma Genetika, melainkan penggabungan antara Algoritma Genetika dengan Algoritma Pencarian Lokal. Struktur Algoritma dalam penelitian tersebut adalah sebagai berikut:
    - Kromosom merepresentasikan calon solusi.
    - *Fitness* merepresentasikan jumlah klausa benar dalam suatu kromosom.
    - GA type merepresentasikan fungsi untuk meneruskan 2 Kromosom dengan nilai *fitness* terbaik ke generasi selanjutnya.
    - *Genetic Operators* merepresentasikan fungsi *crossover* dan *mutation.*
2. Penelitian yang dilakukan oleh Xiangfu Zhao, Liming Zhang, Dantong Ouyang, dan Yu Jiao dengan judul "*Deriving all minimal consistency-based diagnosis sets using SAT solvers*"[13]. Penelitian di atas membuktikan bahwa SAT Solver dapat menyelesaikan uji konsistensi minimal yang merupakan permasalahan NP-*Complete*. Penelitian merepresentasikan permasalahan tersebut dalam bentuk CNF *files*
3. Penelitian yang dilakukan oleh Ahmed A. Abdul Hamed, Medhat A.Tawfeek, Arabi E.Keshk dengan judul "*A genetic algorithm for service flow management with budget constraint in heterogeneous computing*"[14]. Penelitian ini membuktikan bahwa Algoritma genetika dapat menyelesaikan permasalahan manajemen aliran layanan dengan batasan anggaran dalam komputasi heterogen yang merupakan masalah optimasi.
4. Penelitian yang dilakukan oleh T. Hidayat dan A. Bahariyanto Irhasni dengan judul "*SAT Solver dengan DPLL dalam Pemrograman Deklaratif*"[17] dan penelitian yang dilakukan oleh Dicky Puja Pratama dengan judul "*PENYELESAIAN BOOLEAN SATISFIABILITY PROBLEM DENGAN ALGORITMA DAVIS PUTNAM LOGEMANN LOVELAND (DPLL) MENGGUNAKAN JAVA*"[18]. Kedua penelitian di atas mengembangkan SAT Solver dengan Algoritma DPLL dengan bahasa prolog dan bahasa JAVA.

B. Dasar Teori

a. Non Polynomial-Complete

Non Polynomial-Complete (NP-*Complete*) adalah salah satu algoritma yang termasuk dalam *Non Polynomial* (NP). Algoritma Non Polynomial adalah algoritma yang kompleksitas waktu kasus terburuknya tidak dibatasi oleh fungsi polinom dari ukuran masukannya[15]. Sedangkan yang dimaksud dengan NP-*Complete* adalah suatu permasalahan NP yang dapat dikurangi ke dalam waktu polinomial [16] Fungsi polinom sendiri berarti persamaan yang memiliki variabel dengan pangkat bertingkat. Pangkat tertinggi dari sebuah polinomial disebut dengan derajat. Misalnya diberikan persamaan suku banyak x3–x+3, suku banyak tersebut memiliki derajat 3.

b. Formula CNF

Sebuah formula proposisi yang memenuhi syarat sebagai formula normal-form disebut Formula CNF (*Conjunctive Normal Form*). Syarat formula normal-form adalah sebagai berikut[11]:

1. Hanya memiliki operator ∧, ∨, dan ¬.
2. Operator ¬ hanya boleh pada proposisi

Contoh formula-formula CNF (*Conjunctive Normal Form*):

1. $a \wedge b$
2. $a \wedge (b \vee \neg c)$
3. $a \wedge b \wedge (\neg a \vee \neg b)$
4. $(a \vee \neg b) \vee (c \vee \neg d)$
5. $(a \vee \neg b) \wedge b \wedge (\neg a \vee \neg c)$
6. $(a \vee \neg b) \vee (c \vee \neg d) \wedge (b \vee d)$
7. $(a \vee \neg b) \wedge (b \vee \neg c) \wedge (c \vee \neg a)$

c. SAT Problem dan SAT Solver

SAT Problem adalah proses menentukan apakah sebuah formula adalah formula yang *satisfiable* atau *unsatisfiable*.

Sebuah formula dapat dikatakan *satisfiable* jika terdapat kombinasi nilai kebenaran komponen (variabel) pembentuknya yang membuat formula tersebut bernilai benar TRUE dan sebaliknya jika komponen penyusunya membuat formula menjadi FALSE maka disebut *unsatisfiable* [2]. Berikut contohnya:

1. $a \wedge b \wedge (\neg a \vee \neg b)$: tidak *satisfiable*, karena tiap kemungkinan variabel yang dimasukkan menjadikan formula menjadi FALSE
2. $(a \vee b) \wedge b \wedge (a \vee \neg c)$: *satisfiable*, karena jika a = F, b = T, dan c = F maka formula menjadi TRUE.
3. $a \wedge b$: *satisfiable*, karena jika a = T dan b = T maka formula menjadi TRUE.
4. $a \wedge (b \vee \neg c)$: *satisfiable*, karena jika a = T dan c = F maka formula menjadi TRUE.

d. Algoritma Genetika

Algoritma genetika adalah algoritma komputasi yang terinspirasi teori Darwin. Algoritma ini dikembangkan oleh Goldberg dan mengatakan bahwa algoritma genetika adalah komputasi untuk mencari solusi dari suatu permasalahan dengan cara yang lebih alami (*by nature*).

Susunan terkecil dari algoritma genetika adalah karakter, simbol, bilangan ataupun karakter dari sebuah permasalahan. Susunan terkecil ini merupakan sebuah nilai dari gen/alel kumpulan gen/alel merupakan penyusun suatu kromosom. Kumpulan kromosom dinamakan populasi. kromosom sendiri merupakan representasi dari sebuah solusi.

Sebuah generasi terjadi ketika kumpulan Kromosom berevolusi secara berkelanjutan. *Fitness* adalah fungsi untuk mengukur nilai/fungsi objektif. fungsi objektif adalah tingkat keberhasilan solusi terhadap masalah yang ingin diselesaikan dalam tahap evaluasi tiap generasi kromosom. Jika sebuah kromosom punya nilai *fitness* yang tinggi dibandingkan dengan kromosom yang lain pada generasi tersebut. Maka kromosom tersebut akan memiliki peluang lebih besar untuk terpilih lagi pada generasi selanjutnya. Hal tersebut adalah tahapan seleksi.

Pada proses *crossover* terjadi perkawinan antar kromosom-kromosom dalam satu generasi hasil dari perkawinan itu adalah *offspring*. Pada tahap selanjutnya Mutasi terjadi anomali terhadap susunan gen yang merepresentasikan perubahan susunan unsur genetik suatu makhluk hidup akibat adanya faktor alam/seleksi.

Algoritma genetika dapat dinyatakan selesai jika sudah muncul kromosom yang terbaik/memenuhi kriteria/sudah mencapai batas iterasi. Kromosom itulah yang menjadi solusi permasalahan yang ingin diselesaikan.

e. JAVA

JAVA adalah bahasa pemrograman tingkat tinggi yang berorientasi objek. Program JAVA tersusun dari bagian yang disebut kelas. Kelas terdiri atas metode-metode yang melakukan pekerjaan dan mengembalikan informasi setelah melakukan tugasnya. Dalam setiap kelas juga terdapat atribut yang merupakan sifat berupa nilai atau kondisi yang dimiliki kelas tersebut.

f. 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, mengkompilasi, 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.

III. HASIL DAN PEMBAHASAN

Dalam pengembangan makalah tentu perlu suatu metodologi penelitian agar penelitian yang dilakukan berjalan dengan baik. Metodologi penelitian yang digunakan dalam menyusun makalah ini meliputi identifikasi masalah, analisis kebutuhan, pemodelan, implementasi, dan pengujian. Hal ini dilakukan agar dalam penyelesaian tugas akhir lebih mudah dan terarah. Metodologi yang digunakan antara lain:

A. identifikasi masalah

Mengidentifikasi masalah dan kebutuhan, cara kerja dan ruang lingkup aplikasi yang akan dibangun dengan cara:

1. Melakukan analisis terhadap permasalahan SAT Problem dan cara-cara penyelesaiannya.
2. Melakukan analisis terhadap Algoritma Genetika dan implementasinya untuk menyelesaikan permasalahan SAT Problem.
3. Menganalisis penggunaan bahasa JAVA untuk menyelesaikan permasalahan SAT Problem.
4. Membaca literatur baik dari jurnal, buku, ataupun penelitian sebelumnya yang berhubungan penelitian ini.

B. analisis kebutuhan

Pada tahapan ini dilakukan analisis kebutuhan apa saja yang diperlukan untuk membangun aplikasi berbasis bahasa JAVA yang dapat menyelesaikan permasalahan SAT Problem menggunakan Algoritma Genetika dan konsep pendekatan. Pada tahapan ini juga mencakup analisis kebutuhan perangkat lunak dan perangkat keras yang dibutuhkan untuk membangun aplikasi tersebut

C. Pemodelan

Pada tahapan ini dilakukan pemodelan Algoritma Genetika terhadap SAT problem. Berikut ini adalah contoh pemodelan SAT Problem, misal terdapat SAT Problem dengan Formula CNF: (a ∨ b) ∧ (c ∨ ¬d) ∧ (c) ∧ (d)

1. *Inisiasi*

   Langkah pertama adalah menentukan jumlah populasi misalnya ditentukan 4 individu dalam sebuah populasi. Kemudian gen/alelnya akan dibentuk secara acak. Gen/alel SAT Problem memiliki dua kemungkinan yaitu nilai variabel 0 yang merepresentasikan pemberian nilai variabel proposisi *false* atau 1 yang merepresentasikan pemberian nilai variabel proposisi *true*. Lebih lengkapnya adalah sebagai berikut:

   - kromosom[1]=[a;b;c;d]=[0;0;0;0]
   - kromosom[2]=[a;b;c;d]=[1;1;0;0]
   - kromosom[3]=[a;b;c;d]=[0;1;1;1]
   - kromosom[4]=[a;b;c;d]=[0;1;1;0]

2. *Evaluasi Kromosom*

   Evaluasi Kromosom atau penentuan nilai fungsi objektif dapat ditemukan menggunakan rumus tertentu. Dalam penelitian rumus yang digunakan adalah fungsi_objektif_kromosom = jumlah klausa - ((p ∧ q) + (r ∨ ¬s) + (r) + (s)). Lebih lengkapnya adalah sebagai berikut:

   - fungsiObjektif(kromosom[1]) = jumlah klausa - ((0 ∧ 1) + (0 ∨ ¬1) + (0) + (1))
     = 4 – (0 + 1 + 0 + 0)
     = 3
   - fungsiObjektif(kromosom[2]) = jumlah klausa - ((1 ∧ 1) + (0 ∨ ¬1) + (0) + (1))
     = 4 – (1 + 1 + 0 + 1)
     = 3
   - fungsiObjektif(kromosom[3]) = jumlah klausa - ((0 ∧ 1) + (1 ∨ ¬1) + (1) + (1))
     = 4 – (1 + 1 + 1 + 1)
     = 0
   - fungsiObjektif(kromosom[4]) = jumlah klausa - ((0 ∧ 1) + (1 ∨ ¬0) + (1) + (0))
     = 4 – (1 + 1 + 1 + 0)
     = 1

3. *Seleksi Kromosom*

   Pada tahap ini akan ditentukan nilai *fitness*. Nilai tersebut akan menentukan seberapa besar kemungkinan terpilih kromosom tersebut. Untuk mendapatkan nilai *fitness* dalam penelitian ini yang digunakan adalah fungsi *fitness* = (1/(1+fungsiObjektif)), untuk menghindari *error* dari pembagian nol maka fungsi-objektif ditambah dengan satu. Lebih lengkapnya adalah sebagai berikut:

   - fitness[1] = 1 / (fungsiObjektif[1]+1)
     =1/(3+1)
     = 0.25
   - fitness[2] = 1 / (fungsiObjektif[2]+1)
     =1/(3+1)
     = 0.25
   - fitness[3] = 1 / (fungsiObjektif[3]+1)
     =1/ (0+1)
     = 1
   - fitness[4] = 1 / (fungsiObjektif[4]+1)
     =1/(1+1)
     = 0.5
   - totalFitness = 0.25 + 0.25 + 1 + 0.5 = 2

   Setelah itu dilanjutkan dengan proses pencarian probabilitas. Dalam penelitian ini menggunakan rumus: P[i] = fitness[i]/total_fitness. Lebih lengkapnya adalah sebagai berikut:

   - Peluang[1] = 0.25 /2
     = 0.125
   - Peluang[2] = 0. 25/ 2
     = 0.125
   - Peluang[3] = 1/ 2
     = 0.5
   - Peluang[4] = 0.5 / 2
     = 0.25

   Kemudian menentukan kumulatifnya

   - Kumulatif[1] = 0.125
   - Kumulatif [2] = 0.125+ 0.125
     = 0.25
   - Kumulatif [3] = 0.125+ 0.125 + 0.5
     = 0.75
   - Kumulatif [4] = 0.125+ 0.125 + 0.5 +0.25
     = 1

   Kemudian dibangun bilangan secara acak:

   - Acak[1] = 0.56
   - Acak[2] = 0.67
   - Acak[3] = 0.11
   - Acak[4] = 0.82

   Setelah melalui tahap diatas populasi berubah menjadi:

   - kromosom[1]=kromosom[3]
   - kromosom[2]=kromosom[3]
   - kromosom[3]=kromosom[1]
   - kromosom[4]=kromosom[4]

4. *Crossover*

   Pertama bangun bilangan acak R sebanyak jumlah populasi

   - Acak[1] = 0.191
   - Acak[2] = 0.259
   - Acak[3] = 0.760
   - Acak[4] = 0.006

   Berdasarkan bilangan acak tersebut persilangan menjadi:

   - kromosom[1] >< kromosom[4]

- kromosom[4] >< kromosom[1]
-

Kemudian bangun lagi bilangan acak untuk menentukan cut-point:

- Cut[1] = 1
- Cut[2] = 1

offSpring[1] = kromosom[1] >< kromosom[4]
= [0;1;1;1] >< [0;1;1;0]
= [0;1;1;0]
offSpring[4] = kromosom[4] >< kromosom[1]
= [0;1;1;0] >< [0;1;1;1]
= [0;1;1;1]

Sehingga setelah mengalami *crossover* populasi menjadi:

- kromosom [1]= [0;1;1;0]
- kromosom [2]= [0;1;1;1]
- kromosom [3]= [0;0;0;0]
- kromosom [4]= [0;1;1;1]

5. *Mutasi*

Misalkan yang mengalami mutasi adalah kromosom ke-2 gen/alel nomor 2 dan Kromosom ke-3 gen/alel nomor 3. Maka nilai gen/alel pada kromosom tersebut akan berganti dari 0 menjadi 1 dan sebaliknya. Maka populasi kromosom setelah mengalami proses mutasi adalah:

- kromosom[1] = [0;1;1;0]
- kromosom[2] = [0;0;1;1]
- kromosom[3] = [0;0;1;0]
- kromosom[4] = [0;1;1;1]

6. *Perulangan*

Setelah itu proses akan terus berulang sampai jumlah iterasi yang telah ditentukan. kromosom dengan nilai *fitness* terbaik sampai dengan akhir iterasi. Nantinya akan ditawarkan sebagai solusi/pendekatan solusi SAT Problem.

D. *Implementasi*

Pada tahapan ini dilakukan implementasi dari pemodelan yang telah dilakukan pada tahap sebelumnya. Dilakukan pengimplementasian Algoritma Genetika dalam penyelesaian permasalahan SAT Problem dengan konsep pendekatan menggunakan bahasa JAVA.

1. *Pembuatan kelas*

```
//membuat objek demo
SimpleDemoGA demo = new SimpleDemoGA();
```

2. *Inisiasi*

```
//membuat sebuah populasi sebanyak 10 individu
demo.population.initializePopulation(10);

//fungsi untuk mengisi seluruh individu
for(i = 0; i < population.length; i++){
        //fungsi random
        random rn = new random();

        //mengisi individu dengan bilangan random integer 0 sampai 1
        population[i] = rn.nextInt(1);

}
```

3. *Evaluasi Kromosom*

```
//menghitung setiap fitness dari individu
demo.population.calculateFitness();

//fungsi untuk memilih 2 individu terbaik
void selection(){

        fittest = population.getFittest();

        secondFittest = population.getSecondFittest();

}
```

4. *Seleksi Kromosom*

```
//menghitung setiap fitness dari individu
demo.population.calculateFitness();

//fungsi untuk memilih 2 individu terbaik
void selection(){

        fittest = population.getFittest();

        secondFittest = population.getSecondFittest();

}
```

5. *Crossover*

```
//fungsi untuk mengkawinkan 2 individu terbaik
void crossover() {
        //fungsi random
        random rn = new random();


        //memilih secara acak point crossover
        int crossOverPoint = rn.nextInt(population.individuals[0].geneLength);

        //memindahkah nilai antar induk
        for (int i = 0; i < crossOverPoint; i++) {
                int temp = fittest.genes[i];
                fittest.genes[i] = secondFittest.genes[i];
                secondFittest.genes[i] = temp;

        }

}
```

6. *Mutasi*

```
//fungsi untuk melakukan mutasi(kelainan gen)
void mutation() {
        //fungsi random
        Random rn = new Random();

        //memilih point mutasi untuk individu terbaik
        int mutationPoint = rn.nextInt(population.individuals[0].geneLength);

        //ganti nilai dari point yang terkena mutasi
        if (fittest.genes[mutationPoint] == 0) {
                fittest.genes[mutationPoint] = 1;
        } else {
                fittest.genes[mutationPoint] = 0;
        }

        //memilih point mutasi individu terbaik kedua
        mutationPoint = rn.nextInt(population.individuals[0].geneLength);

        //ganti nilai dari point yang terkena mutasi
        if (secondFittest.genes[mutationPoint] == 0) {
                secondFittest.genes[mutationPoint] = 1;
        } else {
                secondFittest.genes[mutationPoint] = 0;
        }
}
```

7. Perulangan

```
//perulangan sebanyak 50 artinya proses diatas akan diulangi sebanyak 50 kali
 while (iteration() < 50) {
    ++demo.generationCount;

    //ulangi seleksi
    demo.selection();

    //ulangi crossover
    demo.crossover();

    //ulangi mutasi dengan probilitas acak untuk terjadi
    if (rn.nextInt()%7 < 5) {
        demo.mutation();
    }
```

E. Pengujian

Dilakukan sebuah pengujian untuk menyelesaikan sebuah SAT Problem. Setiap SAT Problem disimpan dalam sebuah *file* teks (CNF *files*). Format penulisan teks tersebut mengikuti aturan penulisan seperti di Kompetisi SAT (http://satcompetition.org). Pengujian dilakukan dengan menggunakan laptop dengan prosesor i5-7200U @2,5Ghz dan RAM 8 GB.

Permasalahan yang diujikan adalah *sample files*. *Sample files* adalah CNF *files* yang digunakan untuk menilai SAT Solver dengan *resource* relatif kecil. diambil dari web (https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html). Sample files yang digunakan adalah:

No	Nama	Variabel	Klausa
1	simple_v3_c2.cnf	3	2
2	quinn.cnf	16	18

SAT Solver dengan Algoritma Genetika ini berhasil menyelesaikan *sample files* pertama dalam waktu relatif instan/mendekati nol dan menyelesaikan *sample files* kedua dalam waktu 0.75 – 2.33 detik, masing-masing *sample files* dilakukan pengujian sebanyak 10 kali percobaan. Hasil pengujian lebih lengkapnya dapat dilihat pada tabel 1 dan 2.

Tabel 1. Pengujian *sample files* pertama

No	Nama	Waktu
1	simple_v3_c2.cnf	0.021 detik
2	simple_v3_c2.cnf	0.022 detik
3	simple_v3_c2.cnf	0.021 detik
4	simple_v3_c2.cnf	0.025 detik
5	simple_v3_c2.cnf	0.011 detik
6	simple_v3_c2.cnf	0.027 detik
7	simple_v3_c2.cnf	0.019 detik
8	simple_v3_c2.cnf	0.014 detik
9	simple_v3_c2.cnf	0.021 detik
10	simple_v3_c2.cnf	0.021 detik

Tabel 2. Pengujian *sample files* kedua

No	Nama	Waktu
11	quinn.cnf	1.23 detik
12	quinn.cnf	0.75 detik
13	quinn.cnf	2.33 detik
14	quinn.cnf	1.45 detik
15	quinn.cnf	2.26 detik
16	quinn.cnf	1.26 detik
17	quinn.cnf	1.45 detik
18	quinn.cnf	1.34 detik
19	quinn.cnf	2.23 detik
20	quinn.cnf	1.52 detik

IV. KESIMPULAN

Telah berhasil dibentuk sebuah SAT Solver menggunakan bahasa pemrograman JAVA. Algoritma yang digunakan oleh SAT Solver ini adalah Algoritma Genetika selain itu SAT Solver ini juga menggunakan konsep pendekatan. SAT Solver ini mampu menyelesaikan SAT Problem yang diujikan dengan cukup memuaskan karena mampu menyelesaikan SAT Problem tersebut dalam satuan detik.

Namun perlu diketahui bahwa SAT Problem yang diujikan masih sangatlah sederhana. Oleh karena itu perlu penelitian lebih lanjut untuk mengetahui potensi SAT Solver yang sebenarnya.

Pada penelitian selanjutnya, perlu pengembangan lebih lanjut terhadap SAT Solver ini dengan memodifikasi Algoritma Genetika agar lebih efisien dan memodifikasi konsep pendekatan agar mengurangi permasalahan ketepatan mencari solusi.

V. REFERENSI

[1] D. Du, J. Gu and P. M. Pardalos, Satisfiability Problem: Theory and Applications : DIMACS Workshop, March 11-13, 1996, American Mathematical Soc, 1997.

[2] M. Huth and M. Ryan, Logic in Computer Science: Modelling and Reasoning about Systems, New York: Cambridge University Press, 2004.

[3] O. Ohrimenko, P. J. Stuckey and M. Codish, "Propagation = lazy clause generation," *Principles and Practice of Constraint Programming – CP 2007,* p. 544–558, 2007.

[4] R. E. Bryant, S. M. German and M. N. Velev, "Microprocessor Verification Using Efficient Decision Procedures for a Logic of Equality with Uninterpreted Functions," *TABLEAUX '99 Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods ,* pp. 1-13, 1999.

[5] G.-J. Nam, K. A. Sakallah and R. A. Rutenbar, "A new FPGA detailed routing approach via search-based Boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 2002.

[6] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1996.

[7] J. Marques-Silva and K. Sakallah, "GRASP: A New Search Algorithm for Satisfiability," *Proceedings of International Conference on Computer-Aided Design,* pp. 220-227, 1996.

[8] C. Li and Anbulagan, "Heuristics Based on Unit Propagation for Satisfiability Problems," *Proceedings of IJCAI,* pp. 366-371, 1997.

[9] M. Moskewicz, "Chaff: Engineering an Efficient SAT Solver," *39th Design Automation Conference (DAC 2001),* 2001.

[10] Y. Vizel, Y. Weissenbacher and S. Malik, "Boolean Satisfiability Solvers and Their Applications in Model Checking," *Proceedings of the IEEE,* p. 103, 2015.

[11] T. Hidayat, Logika Proposisi, Yogyakarta: Dar Firqin, 2014.

[12] E. Marchiori and C. Rossi, "A Flipping Genetic Algorithm for Hard 3-SAT Problems," *GECCO'99 Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation,* vol. I, pp. 393-400, 1999.

[13] X. Zhao, L. Zhang, D. Ouyang and Y. Jiao, "Deriving all minimal consistency-based diagnosis sets using SAT solvers," *Progress in Natural Science,* vol. 19, no. 4, pp. 489-494, 2009.

[14] A. A. AbdulHamed, M. A. Tawfeek and A. E. Keshk, "A genetic algorithm for service flow management with budget constraint in heterogeneous computing," *Future Computing and Informatics Journal,* vol. 3, no. 2, pp. 341-347, 2018.

[15] O. Darmawan and Y. Nilamsari Kusumastuti, "KONSTRUKSI PELABELAN SISI AJAIB SUPER PADA GRAF ULAT," *BIMASTER,* vol. III, no. 03, 2014.

[16] K. S. and P. R., "Introduction to NP-Complete Problems".

[17] T. Hidayat and A. Bahariyanto Irhasni, "SAT Solver dengan DPLL dalam Pemrograman," 2019.

[18] D. Puja Pratama, "PENYELESAIAN BOOLEAN SATISFIABILITY PROBLEM DENGAN ALGORITMA DAVIS PUTNAM LOGEMANN LOVELAND (DPLL) MENGGUNAKAN JAVA," 2018.