Dokumentasi Tugas 1 Logika Komputasional

1106018871 – Immanuel Rhesa

1106087534 – Victoria Anugrah Lestari

# 1. Cara kerja MiniSat dan SAT Solver pada umumnya

## 1.1 Boolean Constraint Propagation

Boolean Constraint Propagation adalah proses untuk medeteksi unit clauses dan conflict clauses setelah assignment variabel*.* Aturan unit clause berbunyi bahwa apabila semua kecuali satu buah literal pada suatu clause sudah di-*assign* false, maka literal tersebut harus di *assign* true agar membuat formula CNF tersebut satisfied. Apabila pada sebuah clauses semua literalnya false, maka terjadi *conflict clause*  dan perlu dilakukan *conflict clauses analysis.*

## 1.2 VSIDS Decision Heuristic

VSIDS (Variable State Independent Decaying Sum) digunakan untuk menentukan *literal* selanjutnya yang akan di *assign* nilainya. Bobot untuk setiap *literal* ditentukan sesuai dengan banyaknya kemunculan *literal* tersebut dari bentuk CNF yang diberikan. *Literal* dengan bobot tertinggi adalah *literal* yang akan di *assign* selanjutnya. Pada setiap periode, bobot dari setiap *literal* akan dibagi dengan suatu bilangan *constant.* Karena itu, *literal* pada *Clauses* yang baru ditambahkan akan mempunyai bobot / *weight* yang lebih tinggi.

## Conflict Clauses Analysis

*Conflict clause analysis* bertujuan untuk mencari penyebab dari *conflict clause,* yaitu clause yang nilai dari semua literalnya adalah false dan mencoba untuk menyelesaikannya. Metode dari *conflict clauses analysis* ini akan menggunakan Resolution untuk menghasilkan suatu *learn clause* baru. *Learn clause* ini kemudian akan ditambahkan pada daftar *clause* yang tersedia dan kembali dilakukan Boolean Constraint Propagation.

## 1.4 Cara kerja secara keseluruhan

Secara umum, langkah kerja dari algoritma yang dilakukan MiniSat adalah sebagai berikut:

1. Literal yang mempunyai bobot tertinggi akan di-*assign* dengan nilai tertentu. Pada setiap assignment ini, akan ada sebuah Decision Level yang diperlukan sebagai batas *backtracking* pada saat Conflict Clause Analysis*.* Setiap kali *assignment* literal pada langkah ini, maka Decision Level akan bertambah.
2. Kemudian dilakukan Boolean Constraint Propagation untuk setiap unit literal.
3. Bila belum ada Conflict Clause (Clause yang nilai semua literalnya adalah false), ulangi langkah 1 dan 2 hingga semua literal berhasil di-*assign* nilainya, jika semua literal telah berhasil di-*assign,* berarti bentuk CNF tersebut berhasil ditemukan modelnya dan dapat disimpulkan Satisfiability. Apabila terjadi Conflict Clause, maka perlu dilakukan *backtrack*. *Backtrack* dilakukan dengan menghapus assignment dari semua literal pada *decision level* yang tertinggi. Kemudian dilakukan *conflict clause analysis* yang untuk mencari penyebab dari *conflict* tersebut dan mencoba untuk diselesaikan dan dihasilkan suatu *learn clause.* *Learn clause* kemudian ditambahkan kedalam daftar *clause* yang tersedia, dan ulangi kembali langkah 1.

# 2. Problem yang diangkat untuk dipecahkan dengan MiniSat

## 2.1 Map Coloring Problem

Map Coloring Problem adalah permasalahan mewarnai peta di mana daerah yang berbatasan tidak boleh diwarnai dengan warna yang sama. Map Coloring Problem biasa diangkat sebagai contoh dalam materi Constraint Satisfaction Problem (CSP). Akan tetapi, Map Coloring Problem juga dapat diterjemahkan menjadi suatu Boolean Satisfiability Problem sehingga dapat dipecahkan dengan MiniSat.

### 2.1.1 Four Color Theorem

Four Color Theorem menyebutkan bahwa “pada suatu bidang planar yang dipisah-pisahkan sehingga menghasilkan sebuah peta, tidak lebih dari empat warna dibutuhkan untuk mewarnai daerah-daerah pada peta sehingga dua daerah yang bersebelahan tidak berwarna sama.” Syaratnya, sebuah daerah haruslah *contiguous* (tidak boleh terpisah).

### 2.1.2 Encoding Warna

Berdasarkan Four Color Theorem, peta yang dihasilkan pada program oleh user diasumsikan cukup diwarnai menggunakan empat warna saja. Untuk memudahkan proses penyelesaian permasalahan, setiap warna direpresentasikan oleh 2 bit, dengan kemungkinan nilai 0 (false) dan 1 (true):

* 00 – merah
* 01 – kuning
* 10 – hijau
* 11 – biru

### 2.1.3 Representasi Permasalahan

Setiap daerah diberikan 2 buah variabel dan dengan:

* adalah nomor daerah
* adalah bit ke-0 daerah
* adalah bit ke-1 daerah

Daerah yang saling terhubung diilustrasikan oleh sebuah *constraint graph*. Contohnya adalah gambar berikut:

1

2

3

**Gambar 1. Daerah (kiri) dan constraint graph yang berpadanan (kanan)**

Pada contoh di atas, ketiga daerah tidak boleh memiliki warna yang sama. Maka salah satu solusi yang mungkin adalah 1 – merah, 2 – kuning, 3 – biru.

Representasi bitnya adalah , , , , , .

Encoding untuk permasalahan di atas adalah:

|  |
| --- |
|  |

Dengan kata lain, tidak boleh ada kasus di mana dan , serta dan , serta dan .

# 3. Rumusan solusi dari problem yang diangkat, menggunakan notasi/ bahasa MiniSat

## 3.1 Bentuk CNF

Untuk menyederhanakan bentuk encoding yang diperoleh pada nomor sebelumnya, diambil sebuah *case* dimana daerah pertama tidak boleh memiliki warna yang sama dengan daerah ketiga yaitu

|  |
| --- |
|  |

Langkah penyederhanaannya adalah sebagai berikut :

1. Hukum De Morgan

|  |
| --- |
|  |

1. Aturan Distribusi

|  |
| --- |
|  |

Sehingga diperoleh bentuk sederhana seperti berikut

|  |
| --- |
|  |

Dengan encoding yang diperoleh pada nomor sebelumnya, dapat disimpulkan bahwa berapapun banyaknya daerah yang berada pada constraint graph, kalimat logika yang dihasilkan tetap mengikuti pola yang sama. Bentuk kalimat logika untuk seluruh *case* di atas dapat diubah ke bentuk *conjunctive normal form* (CNF) berikut:

|  |
| --- |
|  |

Apabila dirumuskan dengan notasi MiniSat, maka representasinya menjadi:

|  |
| --- |
| p cnf 6 12  -10 -20 -11 -21 0  -10 -20 11 21 0  10 20 -11 -21 0  10 20 11 21 0  -10 -30 -11 -31 0  -10 -30 11 31 0  10 30 -11 -31 0  10 30 11 31 0  -30 -20 -31 -21 0  -30 -20 31 21 0  30 20 -31 -21 0  30 20 31 21 0 |

Jumlah variabel adalah 6 dan jumlah klausa adalah 12. (Variabel pada contoh di atas adalah 10, 11, 20, 21, 30, dan 31.)

## 3.2 Generalisasi masalah

Diberikan suatu *graph* g, konjungsi klausa c merepresentasikan setiap *edge* e yang terdapat pada *graph* g. *Edge* e menghubungkan 2 buah daerah. Untuk setiap daerah i dan j yang terhubung dalam e, konjungsi klausa c akan berbentuk:

|  |
| --- |
|  |

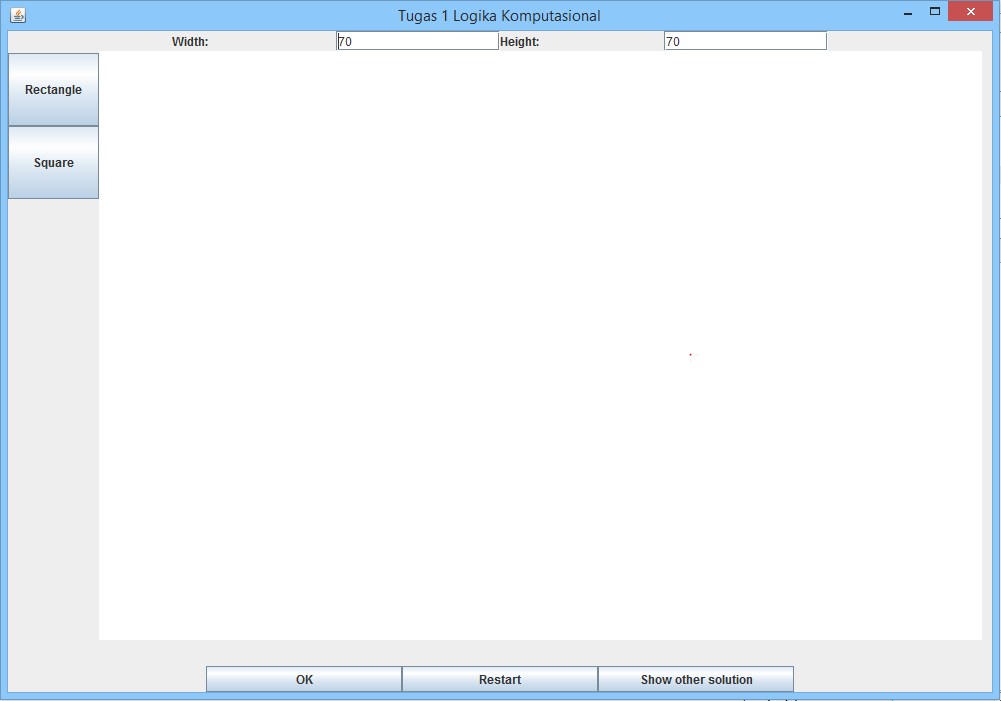
Untuk setiap tambahan daerah yang terhubung (ada di edge), cukup menambahkan sebuah konjungsi klausa c.

# 4. Contoh eksekusi dan outputnya

## 4.1 Contoh eksekusi

Class utama pada program adalah Runner.java. Apabila dijalankan, program akan menampilkan sebuah *graphical user interface* berupa panel dan button. Program akan mensimulasikan *map coloring problem* dengan peta sebagai input dari pengguna. Setiap daerah diwakili oleh sebuah bentuk. Tersedia 2 pilihan bentuk, yaitu persegi dan persegi panjang. Pengguna dapat memilih bentuk dengan mengklik button yang bersesuaian.

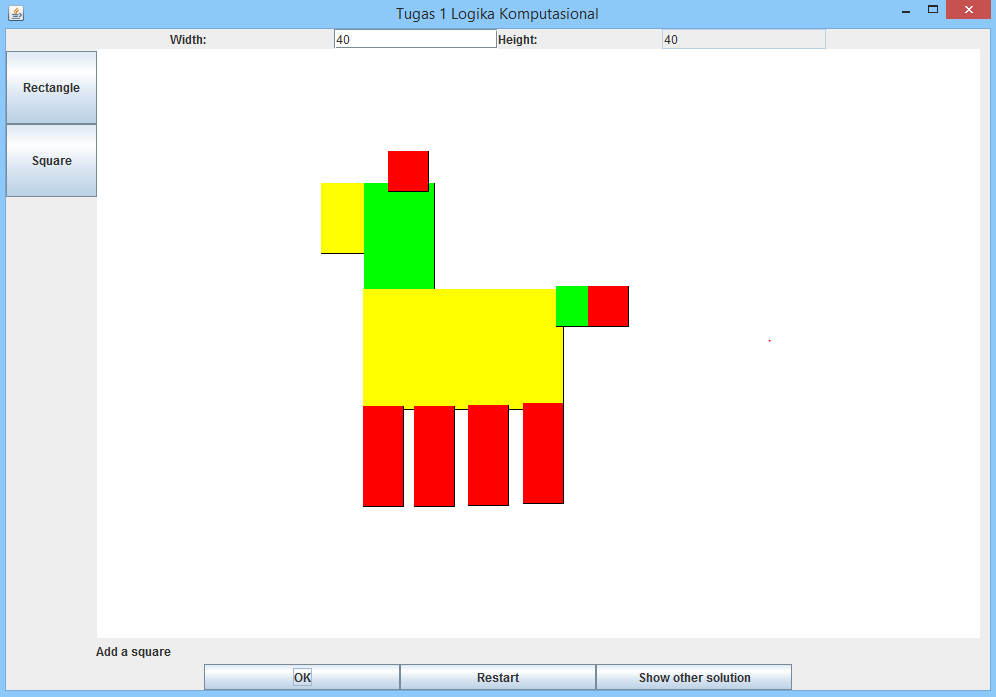
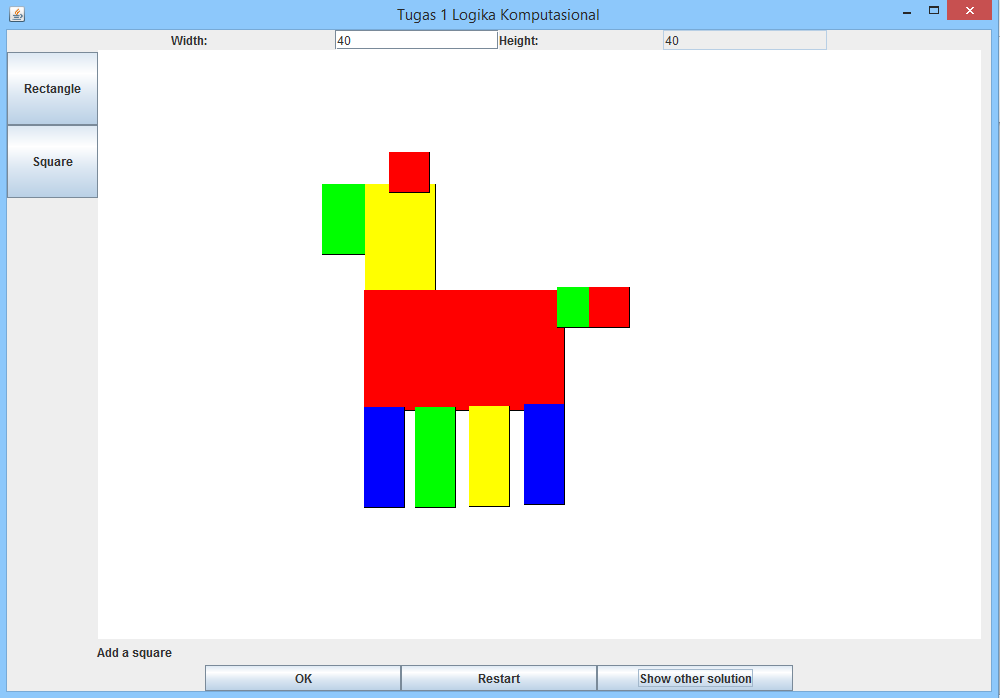
Untuk membuat peta pada panel, pengguna dapat mengklik panel. Ketika pengguna mengklik pada bagian panel yang kosong, sebuah bentuk akan muncul pada panel. Untuk memindahkan bentuk, klik pada area di dalam bentuk tersebut. Dua kali klik (*double click*) pada bentuk akan menghapus bentuk tersebut.



**Gambar 2. Panel kosong ketika program dijalankan**

Pada bagian atas panel terdapat 2 text field, “Width” dan “Height”, untuk mengisi lebar dan panjang persegi panjang. Pada bagian kiri panel terdapat 2 button, yaitu “Square” dan “Rectangle”. Apabila button “Square” diklik, maka daerah yang muncul pada panel berbentuk persegi. Apabila button “Rectangle” diklik, maka daerah yang muncul pada panel berbentuk persegi panjang.

Pada bagian bawah panel terdapat 3 buah button, yaitu button “Ok”, button “Restart”, dan button “Show other solution”. Button “Ok” berfungsi sebagai pemberi solusi pada peta yang telah dibuat pengguna. Button “Restart” berfungsi menghapus semua bentuk yang ada pada panel sehingga pengguna dapat mengulang dari awal. Button “Show other solution” berfungsi menunjukkan solusi lain (pewarnaan lain) pada peta.

**Gambar 3. Daerah pada peta yang sudah diwarnai (kiri) dan contoh solusi lainnya (kanan)**

Output dari program ini adalah daerah-daerah pada peta yang sudah diwarnai. Setiap daerah yang bersebelahan akan memiliki warna berbeda.

## 4.2 Penjelasan program

Program terdiri dari 8 class, yaitu 5 class untuk logic, 2 class untuk GUI, dan 1 class utama yaitu Runner.java.

1. Runner

Class utama pada program. Ketika dijalankan, program akan menampilkan JFrame berisi panel kosong dengan button-button.

1. MainGUI

Class yang bertanggung jawab membuat seluruh GUI program serta logic di belakang tampilan GUI tersebut. MainGUI membuat gambar persegi dan persegi panjang, menjalankan fungsi button, serta memanggil class-class logic untuk menjalankan MiniSat.

1. EncodedColor

Class yang menyimpan 4 buah warna: merah, kuning, hijau, biru, dalam 2 bit. EncodedColor berfungsi sebagai “penerjemah” bit representasi warna (2 buah integer) ke warna aslinya (class Color).

1. Graph

Class yang merupakan penyederhanaan dari graph sebenarnya. Graph ini hanya menyimpan edge list untuk menyimpan keterhubungan suatu daerah dengan daerah lainnya. (Daerah yang tidak terhubung tidak dimasukkan ke dalam graph.)

1. Edge

Class yang merepresentasikan sebuah Edge pada Graph. Class Edge menyimpan 2 buah Integer x dan y yang akan digunakan untuk menyatakan bahwa daerah x terhubung dengan daerah y.

1. CNFMaker

Class yang bertujuan untuk membuat suatu representasi CNF dari sebuah Graph. Representasi CNF direpresentasikan dengan suatu bentuk Array List of Integer yang menyatakan list of clause.

1. MinisatInputMaker

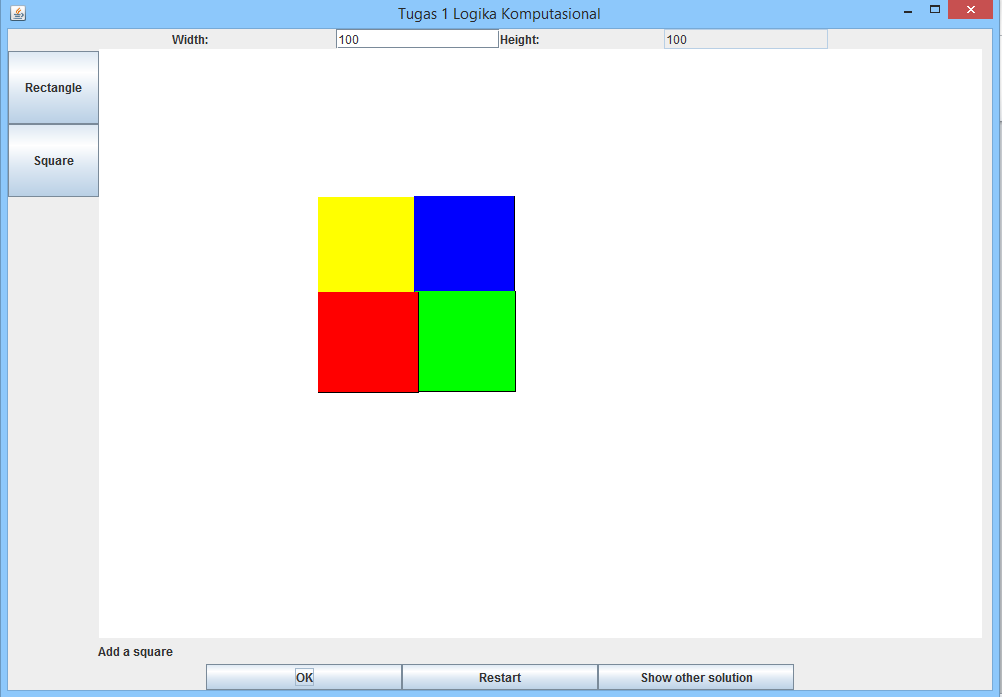
Class yang bertanggung jawab untuk menjalankan fungsi Minisat yang sudah terinstall pada computer dengan menggunakan Java. Class ini akan merepresentasikan list of clause yang berupa array of integer menjadi String input yang sesuai untuk diselesaikan oleh Minisat dan kemudian mengembalikan hasil model yang diperoleh apabila CNF tersebut satisfiability.

1. Minisat

Class yang merupakan *wrapper* Class dari MinisatInputMaker, class ini digunakan untuk menambahkan clause baru kedalam list clause agar kemudian dapat diselesaikan kembali menggunakan minisat. Penambahan clause baru digunakan untuk menambahkan negasi dari clause hasil yang digunakan untuk mencari model penyelesaian yang lain.

## 4.3 Cara kerja program

Misalkan diberikan input peta seperti gambar berikut:



**Gambar 4. Contoh input peta**

Pertama-tama, pengguna membuat persegi pada layar. Persegi kuning adalah nomor 1, biru nomor 2, hijau nomor 3, dan merah nomor 4.

Sistem akan menyimpan seluruh persegi tersebut dalam sebuah list. Saat pengguna menekan tombol “OK”, maka proses SAT solver akan mulai berjalan.

Langkah kerja dari program ini adalah sebagai berikut:

* Program mendeteksi semua persegi yang saling bersentuhan (*intersect*). Kemudian persegi-persegi yang bersentuhan tersebut akan ditambahkan sebagai sebuah Edge pada Graph. Misalkan persegi nomor 1 dan nomor 2 bersentuhan, maka akan dibuat Edge yang menyatakan bahwa persegi 1 dan 2 terhubung. Untuk contoh diatas, maka akan dibuat Edge-edge sebagai berikut {1,2} , {1,3}, {1,4}, {2,3}, {2,4}, {3,4}.
* Dengan fungsi makeCNF, program membentuk representasi CNF dari graph tersebut. Pertama-tama untuk keperluan pewarnaan pada tahap akhir, semua persegi yang terhubung secara terurut akan dimasukkan pada sebuah TreeSet. Untuk mempermudah pembentukan clause CNF, maka setiap persegi yang terhubung tersebut dipetakan pada suatu HashMap dengan value mulai dari 0 hingga k-1 (dimana k merupakan jumlah persegi yang mempunyai pasangan). Untuk contoh kasus diatas, maka dihasilkan Set = {1,2,3,4}, dan HashMap {1 -> 0 , 2->1 , 3->2, 4->3}. Untuk selanjutnya value dari key pada suatu HashMap dinyatakan dengan HM(k) dimana k adalah key pada HashMap. Sehingga HM(2) = 1
* Untuk setiap Edge, akan dibentuk sebuah clause yang menyatakan bentuk Encoding dari permasalahan Map Coloring. Oleh karena itu, setiap persegi perlu dinyatakan sebagai 2 buah literal. Sehingga untuk kasus representasi literal untuk setiap persegi adalah sebagai berikut
* Persegi 1,
* Persegi 2,
* Persegi 3,
* Persegi 4,

Dapat terlihat pola untuk setiap representasi literal yaitu , sedangkan

Setelah didapatkan representasi literal nya, barulah dibentuk clause-clause yang merupakan bentuk generalisasi masalah untuk setiap Edge yang terhubung seperti yang sudah dijelaskan pada bab 3.2. Setiap clause akan direpresentasikan sebagai array of integer dan kumpulan clause dinyatakan dalam ArrayList<int[]>.

Bentuk akhir dari representasi ArrayList untuk contoh kasus diatas adalah:

-1 -3 -2 -4

-1 -3 2 4

1 3 -2 -4

1 3 2 4

-1 -5 -2 -6

-1 -5 2 6

1 5 -2 -6

1 5 2 6

-1 -7 -2 -8

-1 -7 2 8

1 7 -2 -8

1 7 2 8

-3 -5 -4 -6

-3 -5 4 6

3 5 -4 -6

3 5 4 6

-3 -7 -4 -8

-3 -7 4 8

3 7 -4 -8

3 7 4 8

-5 -7 -6 -8

-5 -7 6 8

5 7 -6 -8

5 7 6 8

* Setelah bentuk CNF berhasil dibuat, program akan mengubah bentuk CNF tersebut menjadi String yang siap di proses oleh minisat. Kemudian perintah minisat dijalankan dari dalam program Java.
* Hasil dari eksekusi minisat akan menghasilkan array of integer yang merupakan model penyelesaian dari bentuk CNF yang diberikan (jika ada).
* Untuk contoh diatas, maka hasil minisat akan menghasilkan model { -1, 2,3,4,5,-6,-7,-8}.
* Dengan model tersebut, maka persegi 1 akan diwarnai dengan encoding 01, persegi 2 dengan 11, persegi 3 dengan 10, sedangkan persegi 4 dengan 00.

# Referensi

<http://minisat.se/downloads/escar05.pdf>

<http://minisat.se/downloads/synth_in_sat.pdf>

<http://cacm.acm.org/magazines/2009/8/34498-boolean-satisfiability-from-theoretical-hardness-to-practical-success/fulltext>

<http://mathworld.wolfram.com/Four-ColorTheorem.html>

<http://www.c3.lanl.gov/mega-math/gloss/math/4ct.html>

http://swtv.kaist.ac.kr/lab-orientation/Intro%20to%20MiniSat%20v1.14.pptx