SATISFIABILITY METHODS FOR COLOURING GRAPHS

Munmun Dey[1] and Amitava Bagchi[2]

[1]Department of Computer Science and Engineering, Sanaka Educational Trust's Group of Institutions, Durgapur, West Bengal, India
86munmun@gmail.com
[2]Department of Computer Science and Engineering, Heritage Institute of Technology, West Bengal, India
amitava.bagchi@heritage.edu.in

ABSTRACT

The graph colouring problem can be solved using methods based on Satisfiability (SAT). An instance of the problem is defined by a Boolean expression written using Boolean variables and the logical connectives AND, OR and NOT. It has to be determined whether there is an assignment of TRUE and FALSE values to the variables that makes the entire expression true.A SAT problem is syntactically and semantically quite simple. Many Constraint Satisfaction Problems (CSPs)in AI and OR can be formulated in SAT. These make use of two kinds of searchalgorithms: Deterministic and Randomized.It has been found that deterministic methods when run on hard CSP instances are frequently very slow in execution.A deterministic method always outputs a solution in the end, but it can take an enormous amount of time to do so.This has led to the development of randomized search algorithms like GSAT, which are typically based on local (i.e., neighbourhood) search. Such methodshave been applied very successfully to find good solutions to hard decision problems.

KEYWORD

SAT, GSAT, Graph Colouring, Randomized Search Algorithms, CSPs

1. INTRODUCTION

Many problems in Artificial Intelligence (AI) and Operations Research (OR) can be formulated as Constraint Satisfaction Problems (CSPs). In a CSP, there is a set of variables and a set of constraints. The variables must be assigned values from specified domains in such a way that all the given constraints are satisfied. An assignment of values to the variables that satisfies all the constraints yields a feasible solution. The objective is to find one feasible solution, or in some cases, all feasible solutions. Sometimes, it is possible to associate a measure of quality with a feasible solution. In such cases the objective might be to find the feasible solution of highest quality.

Decision problems such as the Graph Colouring Problem and the Satisfiability Problem (SAT) can also be viewed as CSPs. The Satisfiability Problem is particularly interesting because it can be used as a stepping stone for solving other decision problems. Problem instances from domains such as Graph Colouring can be encoded into SAT and then solved by the help of SAT algorithms.

Rupak Bhattacharyya et al. (Eds) : ACER 2013,
pp. 135–147, 2013. © CS & IT-CSCP 2013 DOI : 10.5121/csit.2013.3213

The work reported in this report is organized in the remaining chapters as follows. In Chapter 2 we firstexplain graph colouring, then show how a graphcolouringproblem can be formulated and solved in SAT using a deterministic approach. We also list the program code andthe output.In Chapter 3we describe how the graphcolouringproblem can be formulated in the GSAT framework, and show how it can be solved using GSAT with a tabulist. Chapter 4lists the GSAT Experimental Results. Chapter 5 summarizes the report

2. GRAPH COLOURING

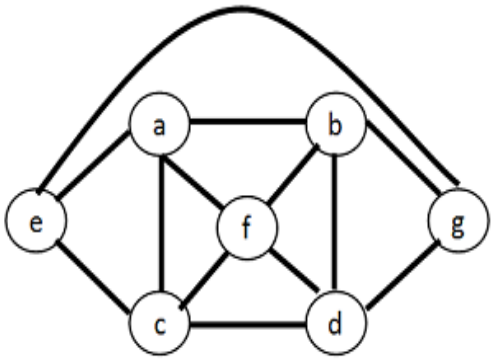
Here only undirected graphs are considered. To colour a graph G means to assigns a colour to each vertex of G with the restriction that two adjacent vertices are not the same colour.

2.1 Graph Colouring Using SAT

This method is based on propositional logic. Let A = $\{a_1, a_2,..., a_n\}$ be a set of $n \geq 1$ Boolean variables. If a is a variable in A then a and ~a are called *literals* over A. A *truth assignment* for A is a total function t: A $\rightarrow$ {*true*, *false*}. Under any truth assignment, *ais true* if and only if ~a is *false*, and a and ~~a have identical truth-values. A *clause* C over A is a set of literals over A. It represents the disjunction of the literals, and is *valid* (or *satisfied*) under a truth assignment if and only if at least one of its literals is *true* under the assignment. It is *invalid* (or *unsatisfied*) under the truth assignment if *every* literal in it is *false* under the assignment. A set C of clauses over A is *satisfiable* if there exist a truth assignment for A such that every clause in C is valid under the assignment. In the Satisfiability Problem (SAT) we are required to determine whether a given set C of clauses is satisfiable. In SAT the clauses represent the constraints to be satisfied when assigning truth-values to the Boolean variables. In SAT, the number of literals in a clause can vary.

2.1.1 SAT formulation

In the SAT formulation there are three types of clauses. A Type 1 clause states that two adjacent nodes cannot have the same colour. A Type 2 clause states that each node must be assigned at least one of the available s colours. There is an Extended Formulation that has Type 3 clauses in addition to Type 1 and Type 2 clauses. A Type 3 clause states that a node can be assigned at most one of the colours.



2.1.2 Example

Table 1: Adjacency matrix

0	1	1	0	1	1	0
1	0	0	1	0	1	1
1	0	0	1	1	1	0
0	1	1	0	0	1	1
1	0	1	0	0	0	1
1	1	1	1	0	0	0
0	1	0	1	1	0	0

Figure 1:Example of an Undirected Graph

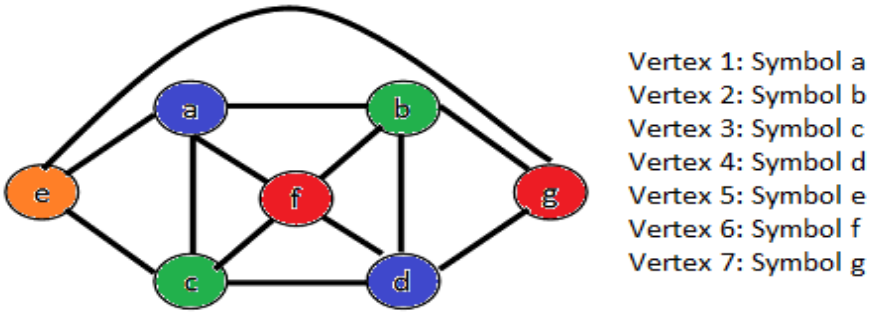


Figure 2: Result of the possible colouring of the above graph.

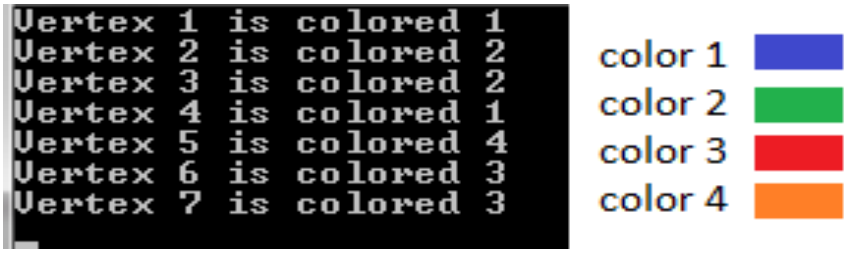


Figure 3: Screenshot of calculating minimum colour needed to colour the above graph.

2.2 GraphColouring Problem Using SAT framework

Graph Coloring Problem can be formulated in the SAT framework. The solution steps are as follows:

Step 1. INPUT: a) Number of nodes.
b)Number of edges.
c)Number of colours.

Step 2: Identification of variables

Step 3: Construction of the clauses

Step 4: Assignment of truth values to literals

*Step 5:*Checking that each clauses is valid.

Step 6: Steps 4 and 5 are repeated until all the clauses are true.

2.2.1 Example

This example will show how a graph coloring problem can be solved using SAT.

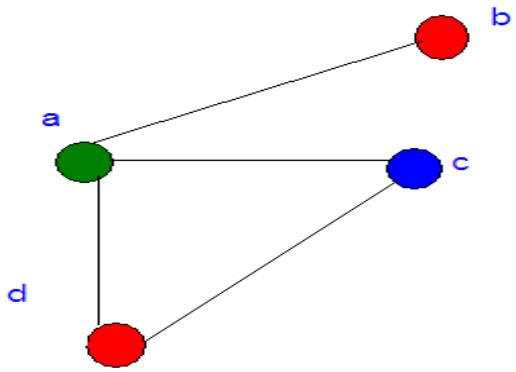


Figure 4: Example of an undirected graph with colour

Figure 4 shows a simple graph consisting of 4 nodes and 4 edges. Suppose we have to determine whether it can be colored with 3 colors named 1, 2 and 3.*a1* is true when a is coloured with colour 1 and is false otherwise .We have seen that constraints here are represented by clauses. The constraint that two adjacent clauses cannot be colored with the same color can be expressed by the following three

Type 1 clauses: ~a1 ∨ ~c1 ~a2 ∨ ~c2 ~a3 ∨ ~c3

As there are 4 edges in total, and there are three Type 1 clauses corresponding to each edge, the total number of such clauses will be **12**.

The constraint that a node must not be left uncolored can be expressed in case of node 1 by the following

Type 2 clause: a1 ∨ a2 ∨ a3

For four nodes in the graph the total number of Type 2 clauses will be **4**.
Type 3 clauses force a node such as node to be colored by a single color. The clauses are as follows:

Type 3 clauses: ~a1 ∨ ~a2 ~a1 ∨ ~a3 ~a3∨ ~a2

The number of Type 3 clauses for the graph of Figure 1 will be **12**, as there are four nodes and three colours. Thus the total number of clauses in the graph will be 28 in this case.

2.2.2 Example

We illustrate below how SAT solves a typical satisfiability problem.

Problem:*Coloringthe graph G1 shown below with 3 colours.*

Table 2: Adjacency matrix

0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0

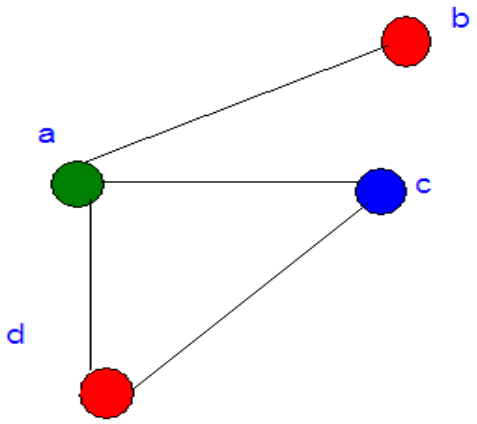


Figure 5:Example of an undirected graph G1

n =4 (n=Number of nodes)
e=4 (e= Number of edges)
c=3 (c= Number of colours)
Variables
a1, ~a1, a2, ~a2, a3, ~a3, b1, ~b1,b2, ~b2, b3, ~b3,c1, ~c1, c2, ~c2, c3, ~c3, d1,~d1,d2, ~d2, d3, ~d3.
Clauses:

Table 3: Clauses

Sl No.	Clauses	Sl No.	Clauses	Sl No.	Clauses	Sl No.	Clauses
1	~a1v~c1	**8**	~d2v~c2	**15**	c1vc2vc3	**22**	~b2v~b3
2	~a2v~c2	**9**	~d1v~c1	**16**	d1vd2vd3	**23**	~c1v~c2
3	~a3v~c3	**10**	~a1v~b1	**17**	~a1v~a2	**24**	~c1v~c3
4	~a1v~d1	**11**	~a2v~b2	**18**	~a1v~a3	**25**	~c2v~c3
5	~a2v~d2	**12**	~a3v~b3	**19**	~a3v~a2	**26**	~d1v~d2
6	~a3v~d3	**13**	a1va2va3	**20**	~b1v~b2	**27**	~d1v~d3
7	~d3v~c3	**14**	b1vb2vb3	**21**	~b1v~b3	**28**	~d2v~d3

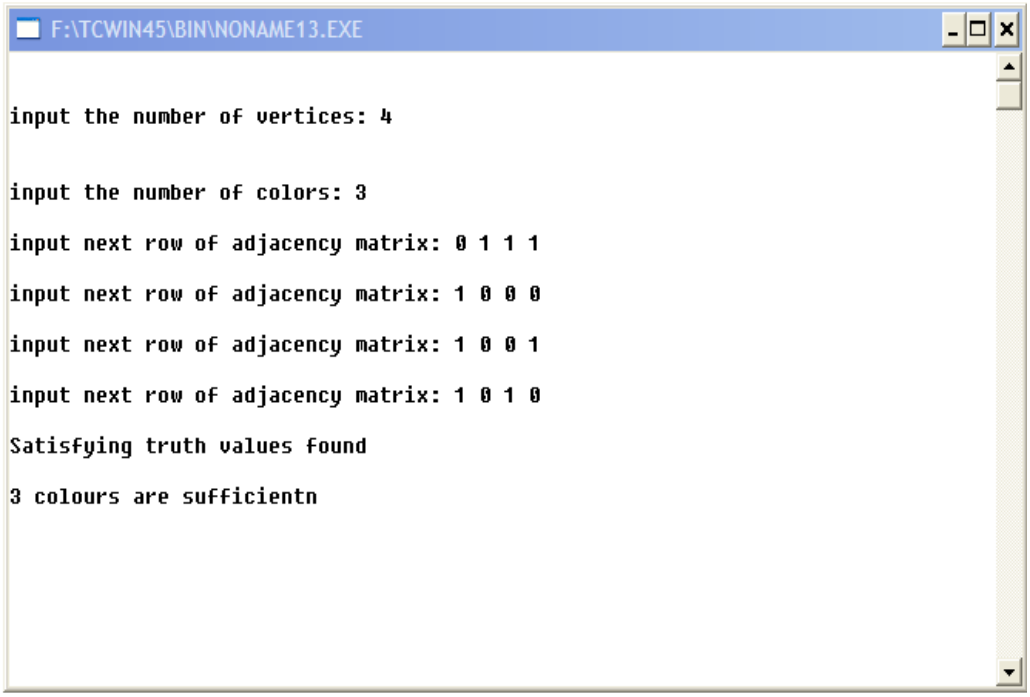


Figure 6:Screenshot of result for the above example

Problem:*Colouring Graph G2 With 3 colours.*

Figure7: Example of an undirected graph G2

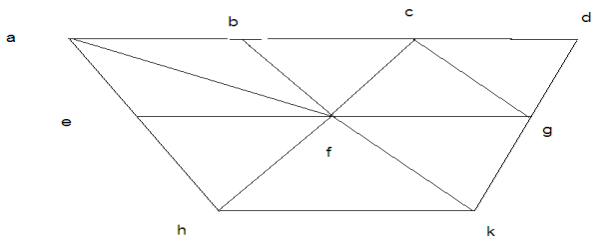


Table 4: Adjacency matrix

0	1	0	0	1	1	0	0	0
1	0	1	0	0	1	0	0	0
0	1	0	1	0	1	1	0	0
0	0	1	0	0	0	1	0	0
1	0	0	0	0	1	0	1	0
1	1	1	0	1	0	1	1	1
0	0	1	1	0	1	0	0	1
0	0	0	0	1	1	0	0	1
0	0	0	0	0	1	1	1	0

n=9 (n=Number of nodes**)**
e=16 (e= Number of edges**)**
c=3(c= Number of colours**)**

Number of Variables:
a1, ~a1, a2, ~a2, a3, ~a3, b1, ~b1 ,b2, ~b2, b3, ~b3 ,c1, ~c1, c2, ~c2, c3, ~c3, d1,~d1 , d2, ~d2, d3, ~d3 , e1, ~e1, e2, ~e2, e3, ~e3 , f1, ~f1, f2, ~f2, f3, ~f3, g1, ~g1, g2, ~g2, g3, ~g3, h1, ~h1, h2, ~h2, h3, ~h3, k1, ~k1, k2, ~k2, k3, ~k3

Number of Clauses:

Table 5: Number of Clauses

Sl No.	Clauses	Sl No.	Clauses	Sl No.	Clauses	Sl No.	Clauses
1	~a1v~b2	**20**	~c2v~g2	**39**	~f3v~h3	**58**	~a1v~a2
2	~a2v~b2	**21**	~c3v~g3	**40**	~f1v~k1	**59**	~a1v~a3
3	~a3v~b3	**22**	~d1v~g1	**41**	~f2v~k2	**60**	~a3v~a2
4	~b1v~c1	**23**	~d2v~g2	**42**	~f3v~k3	**61**	~b1v~b2
5	~b2v~c2	**24**	~d3v~g3	**43**	~g1v~k1	**62**	~b1v~b3
6	~b3v~c3	**25**	~a1v~e1	**44**	~g2v~k2	**63**	~b3v~b2
7	~c1v~d1	**26**	~a2v~e2	**45**	~g3v~k3	**64**	~c1v~c2
8	~c2v~d2	**27**	~a3v~e3	**46**	~h1v~k1	**65**	~c1v~c3
9	~c3v~d3	**28**	~e1v~f1	**47**	~h2v~k2	**66**	~c2v~c3
10	~a1v~f1	**29**	~e2v~f2	**48**	~h3v~k3	**67**	~d1v~d2
11	~a2v~f2	**30**	~e3v~f3	**49**	a1va2va3	**68**	~d1v~d3
12	~a3v~f3	**31**	~f1v~g1	**50**	b1vb2vb3	**69**	~d3v~d2
13	~b1v~f1	**32**	~f2v~g2	**51**	c1vc2vc3	**70**	~e1v~e2
14	~b2v~f2	**33**	~f3v~g3	**52**	d1vd2vd3	**71**	~e1v~e3
15	~b3v~f3	**34**	~e1v~h1	**53**	e1ve2ve3	**72**	~e2v~e3
16	~c1v~f1	**35**	~e2v~h2	**54**	f1vf2vf3	**73**	~f1v~f2
17	~c2v~f2	**36**	~e3v~h3	**55**	g1vg2vg3	**74**	~f1v~f3
18	~c3v~f3	**37**	~f1v~h1	**56**	h1vh2vh3	**75**	~f2v~f3
19	~c1v~g1	**38**	~f2v~h2	**57**	k1vk2vk3		

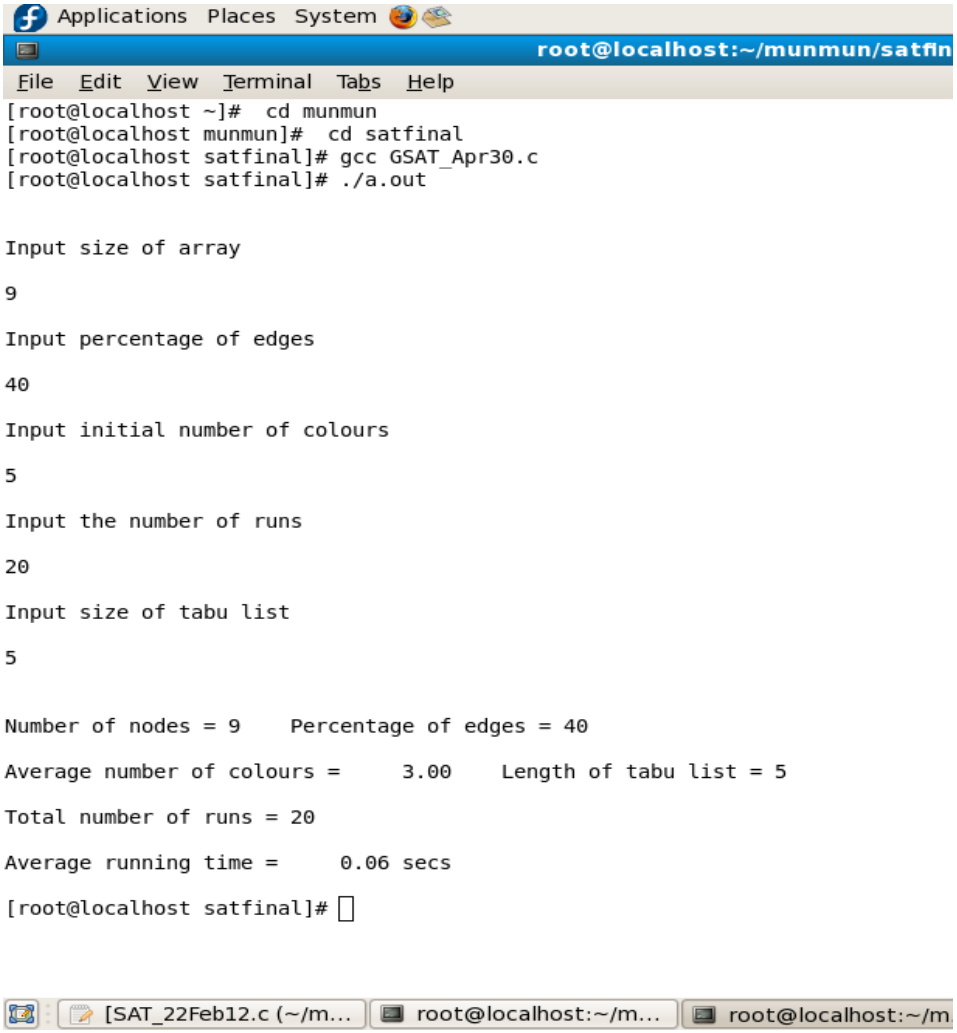


Figure 8:Screenshot of result for the above example

3. RANDOMIZED SEARCH ALGORITHMS

Deterministic search methods frequently run slowly on hard CSP instances. This has led to the formulation and development of randomized search algorithms. Most randomized algorithms are based on local search methods and are therefore often called *Local Search Algorithms*. These can find near-optimal solutions. A local search method generally initiates the search process in some randomly chosen point (or set of points) in the *search space* (sometimes called the *solution space*). Such a point is often called a *candidatesolution.* A *feasible* (or complete) solution consists of a complete set of assignments of values to the search variables. The set of points reachable from a point by making all possible moves is called the *neighborhood* of the point. A local search method has to evaluate the neighborhood points and determine to which neighbor the next move should be made. The evaluation is often accomplished with the aid of an objective function. This function is generally problem dependent and rates the likelihood that the point leads closer to a feasible solution. For example, the objective function for the algorithm GSAT for any problem instance formulated as a Satisfiability Problem is typically taken as the total number of unsatisfied clauses. Auxiliary data structures such as tab lists can help to improve the performance of the algorithm. The initial solution

can be chosen either randomly or by the use of a heuristic. The search then proceeds to a neighboringpoint and from that point to another neighboring point, and so on in an iterative way**.** Although moves are made in a systematic manner, local search algorithms are typically *incomplete.* Local search algorithms often make use of stochastic mechanisms.

3.1 Local Search Algorithms

In this section, however, we are concerned only with general local search methods such as GSAT and Tabu Search.

3.2.1GSAT

The greedy local search procedure GSAT assigns truth values to variables in an effort to satisfy all the clauses in a given set. At start, the procedure randomly assigns truth-values to variables. This amounts to picking a solution randomly from the solution space. Typically this truth assignment fails to satisfy all the clauses. If the truth-value of a variable is *flipped* (*i.e*., complemented), the total number of satisfied clauses will change. Suppose a variable v is flipped. Then the unsatisfied clauses where v occurs get satisfied. The number of such clauses is represented by *make(v)*. At the same time, before flipping v, there were some clauses satisfied only by the variable v. As v is flipped these clauses get unsatisfied. The number of such clauses is given by *break(v)*. The difference make(v) - break(v) is the net increase in the number of satisfied clauses and is referred to as *gain(v)*. Flipping a variable means moving to a neighboring point in the solution space. We have mentioned above that an objective function can help to guide us where to move next in the neighborhood. In GSAT the value of the function is the number of unsatisfied clauses. GSAT looks for the variable with the property that its truth-value when flipped causes the *largest net decrease* in the number of unsatisfied clauses. It flips the truth-value of this variable, and again looks for such a variable. Ties are resolved arbitrarily. This is repeated until a satisfying assignment is found. If no satisfying assignment is found within a specified number of flips (*maxflips*), the procedure is restarted with a new initial truth assignment. This cycle is repeated a specified number of times (*maxtries*). The algorithm always tries to select the most promising neighbour at each step. It is therefore called a *greedy* local search algorithm.

3.3Graph Colouring Problem Using GSAT framework

The solution steps are therefore as follows:

Step1:INPUTa) Size of array

b) Percentage of edges
c)Initial number of colours
d) The number of runs
e)Size of tabu list

Step 2: Creation of list of literals.
Step 3: Construct the clauses
Step 4: Assignrandom truth assignment value to literals
Step 5: Check whether that each clauses is valid.
Step 6: If this truth assignment does not satisfy all the clauses
Step 7: Determine the effect of flipping the truth-value of each variable
Step 8: Count the satisfied clauses after the bit is changed
Step 9: Determine the variables flipping which cause the maximumdecrease in the number of Unsatisfied clauses;

Select a variable from these set resolving ties arbitrarily;
Flip the selected variable.
Step 10: Repeat Step 4 and Step 5until all the clauses are true**.**

3.4Procedure GSAT

Anillustrationis done below how GSAT solves a typical satisfiability problem. Consider the following set of six clauses:

C_1:	**a_3**	**U**	**a_4**		
C_2:	**a_5**	**U**	**~a_4**		
C_3:	**a_1**	**U**	**a_2**	**U**	**~a_3**
C_4:	**~a_1**	**U**	**~a_3**	**U**	**~a_5**
C_5:	**~a_2**	**U**	**~a_3**		
C_6:	**a_3**	**U**	**~a_4**		

There are five variables a_1, a_2, a_3, a_4 and a_5. Suppose the initial random truth assignment is as follows:

a_1 = true,a_2 = false,a_3 = true, a_4 = false, a_5 = true.

This truth assignment does not satisfy all the clauses; in particular, clause C_4 remains unsatisfied. We must now determine the effect of flipping the truth-value of each variable. Suppose that when a variable is flipped, make clauses that are currently unsatisfied get satisfied, and breakclauses that are currently satisfied become unsatisfied. The overall decrease in the number of unsatisfied clauses is then given by the expression **gain = make – break**. We have to find the variable with the largest gain. Here

gain(x_1) = 1 – 1 =0,
gain(x_2) = 0 – 1 = -1,
gain(x_3) = 1 – 1 = 0,
gain(x_4) = 0 – 0 = 0,
gain(x_5) = 1 – 0 = 1.

We change the truth value of x_5 to false since this results in the largest gain, and find as a result that all the clauses are satisfied.

3.5 Tabu Search Strategy

GSAT can significantly improved by tabu search strategy to ensure that the same variable is not flipped again and again.Tabulist is initially empty and is implemented as a FIFO queue.The variable that has just been flipped is inserted into the list.The variable to be flipped next is selected randomly from among those variables not in the tabulist.Thus some variables are prevented from being flipped for a limited period of time,determine by the length of the tabulist. Tabu list in GSAT not only helped to reduce the running time,it helped us to solve some problems that could not be solved at all without using tabulist.

This chapter describes how randomized algorithms perform local searches. In spite of being incomplete in nature, a local search method is often preferred over a deterministic search method when a near-optimal solution has to be found in real time.GSAT algorithms based on local search have been described in this chapter. These include Tabu Search. The Satisfiability Problem (SAT) is explained. The most interesting property of the Satisfiability such as Graph Colouring can be encoded into SAT has been illustrated. Once encoded into SAT, the problem can be solved by the

use of SAT algorithms and Graph Colouring can be encoded into GSAT and Tabu search has been illustrated with experimental result.

4. EXPERIMENTAL RESULTS

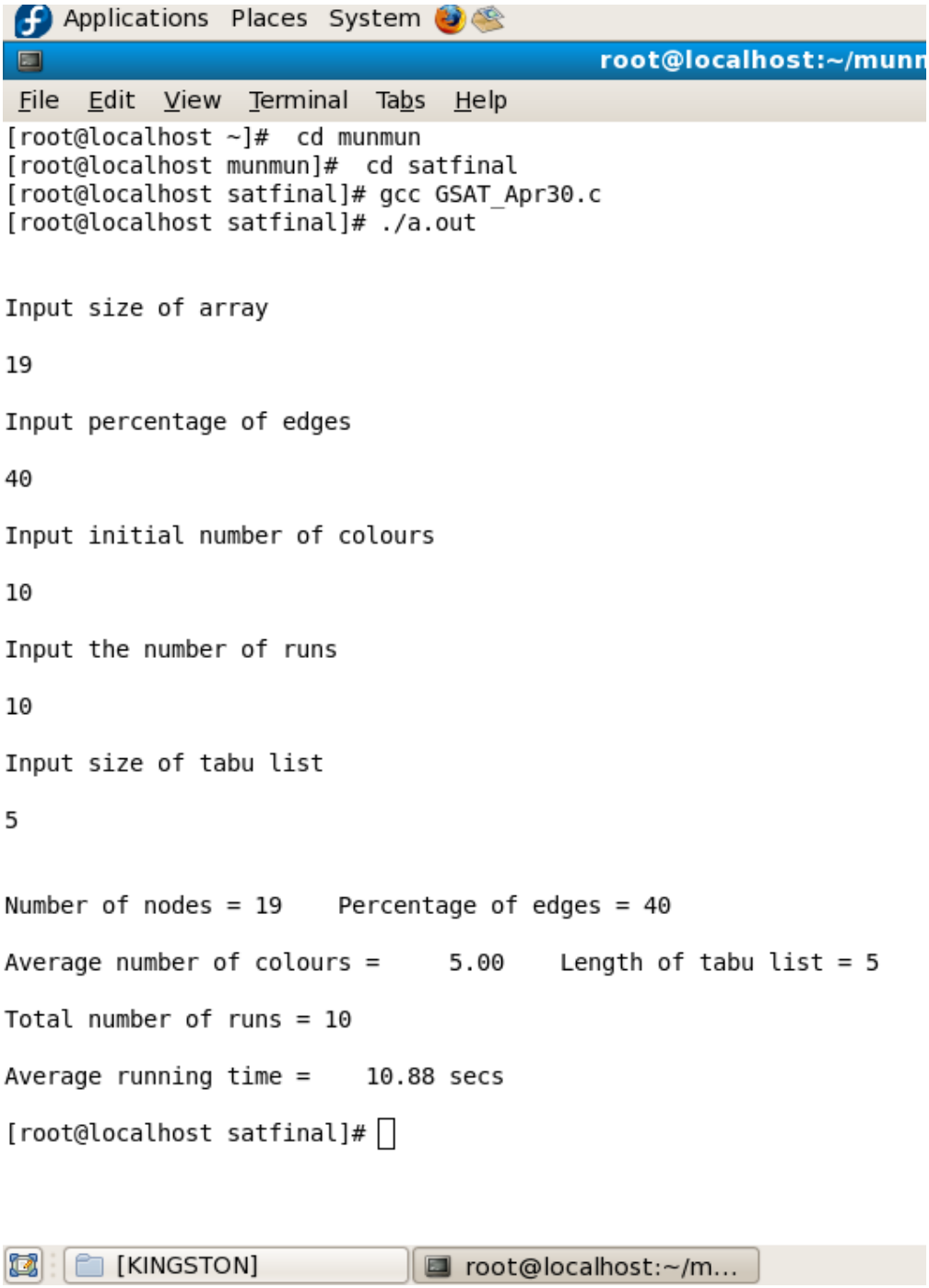


Figure 9:Screenshot of result of the Graph Colouring Problem using GSAT with low time complexity

No.	Size of Array	Percentage of Edges	Initial Number of Colours	Number of Runs	Size of Tabulist	Average Number of Colours	Average Running Time (secs)
1.	9	50	5	20	0	4.00	0.17
2.	9	50	5	20	1	4.00	0.17
3.	9	50	5	20	2	4.00	0.17
4.	9	50	5	20	3	4.00	0.17
5.	9	50	5	20	4	4.00	0.16
6.	9	50	5	20	5	4.00	0.16
7.	9	50	5	20	6	4.00	0.16
8.	9	50	5	20	7	4.00	0.16
9.	10	50	6	20	0	4.00	0.24
10.	10	50	6	20	1	4.00	0.24
11.	10	50	6	20	2	4.00	0.24
12.	10	50	6	20	3	4.00	0.24
13	10	50	6	20	4	4.00	0.24
14	10	50	6	20	5	4.00	0.24
15.	11	50	6	20	0	5.00	0.91
16.	11	50	6	20	3	5.00	0.92
17.	12	50	6	20	0	4.00	0.49
18.	12	50	6	20	3	4.00	0.49
19.	15	50	6	20	0	5.00	2.86
20.	15	50	6	20	3	5.00	2.86
21.	18	50	8	20	0	5.00	5.65
22.	20	50	8	20	0	6.00	17.54

Table 6: Colouring Random Graph with Tabulist

5. CONCLUSION

In this paper both theoretical and experimental studies are done successfully to calculate the performance of SAT for a Graph coloring problem. The role played by SAT as an intermediate domain for solving problems. SAT is not sufficient to calculate the minimum number of color needed to color a graph.SAT technique only extracted that input number of color can sufficient to color the graph or not. Hera also shows how this problem can be encoded in SAT. Finally, **GSAT** method is introduced to use for solving graph coloring problems. GSAT method is enabled to calculate the number of color need to color a graph. This paper also established a new algorithm using GSAT which can satisfy the problem with low time complexity.

REFERENCES

[1] Acharyya S, TheSatisfiability Problem: A Constraint Satisfaction Approach, Ph D Thesis, Computer Science & Engineering, University of Calcutta, 2001

[2] Acharyya S, SAT Algorithms for Colouring Some Special Classes of Graphs: Some Theoretical and Experimental Results, Journal on Satisfiability, Boolean Modeling and Computation(JSAT),vol 4, no 1, pp 33-55, 2007

[3] Acharyya S &Bagchi A, Local search methods for coloring graphs: Role of tabu list, Artificial Intelligence: Emerging Trends & Applications (Proc KBCS-2004), International Conf on Knowledge-Based Computer Systems, Hyderabad, India, Dec 2004, pp 462-472

[4] Selman B,Levesque H J & Mitchell D J, A New Method for Solving Hard SatisfiabilityProblems, Proc AAAI 92, American Association for Artificial Intelligence, 1992, pp 440 446, 1992

AUTHORS

MunmunDey is an Assistant Professorat Sanaka Educational Trust's Group of Institutions, Durgapur, West Bengal, India since 2012. She received his M.E degree from Heritage Institute of Technology in year 2012 and B-Tech degree from Meghnad Saha Institute of Technology in 2009. Her research interests include the field of Optimization techniques.

Prof. (Dr.) Amitava Bagchi is a Professor at Heritage Institute of Technology, West Bengal, India. He received his DSc. degree from MIT, USA. Prof Bagchi has 43 years teaching and Industry experiences. His research interests include the field of Optimization techniques. He has about 55 referred national and international publications to her credit.