SAT Solving Using an Epistasis Reducer Algorithm plus a GA

Eduardo Rodriguez-Tello and Jose Torres-Jimenez
ITESM Campus Cuernavaca, Computer Science Department
Av. Paseo de la Reforma 182-A. Lomas de Cuernavaca
62589, Temixco Morelos, MEXICO
{ertello, jtj}@itesm.mx

Abstract

A novel method, for solving satisfiability (SAT) instances is presented. It is based on two components: a) An Epistasis Reducer Algorithm (ERA) that produces a more suited representation (with lower epistasis) for a Genetic Algorithm (GA) by preprocessing the original SAT problem; and b) A Genetic Algorithm that solves the preprocessed instances. ERA is implemented by a simulated annealing algorithm (SA), which transforms the original SAT problem by rearranging the variables to satisfy the condition that the most related ones are in closer positions inside the chromosome. Results of experimentation demonstrated that the proposed combined approach outperforms GA in all the tests accomplished.

1. Introduction

A genetic algorithm (GA) is based on three elements: an internal representation, an external evaluation function and an evolutionary mechanism. It is well known that in any GA the choice of the internal representation greatly conditions its performance [4][14]. The representation must be designed in such a way that the gene-interaction (*epistasis*) is kept as low as possible. It is also advantageous to arrange the coding so that *building blocks* may form to aid the convergence process towards the global optimum.

We are specially interested in applying GAs to the satisfiability (SAT) problem. The most general statement of the SAT problem is very simple. Given a well-formed boolean formula *F* in its conjunctive normal form (CNF), is there a truth assignment for the literals that satisfies it? SAT, is of great importance in computer science; in theory is one of the six basic core NP-complete problems [9]; in practice many applications can be formulated and solved with SAT [11] [1] [6].

Even though the SAT problem is NP-complete, many methods have been developed to solve it. These methods can be classified into two large categories: complete and incomplete methods. Theoretically, complete methods are able to find the solution, or to prove that no solution exists provided that no time constraint is present. However, the combinatorial nature of the problem makes these complete methods impractical when the size of the problem increases. In contrast incomplete methods are based on efficient heuristics, which help to find sub-optimal solutions when applied to large optimization problems. This category includes such methods as simulated annealing [15], local search [13], and genetic algorithms (GAs) [8].

SAT has a natural representation in GAs: binary strings of length *N* in which the *i-th* bit represents the truth value of the *i-th* boolean variable present in the SAT formula. It is hard to imagine a better representation for this problem. Surprisingly, no efficient genetic algorithm has been found yet using this representation. We have strong reasons to think that this poor performance is caused by the effect of epistasis. It is reported in [7] that generation of building blocks will not occur when epistasis in the representation of a problem is high, in other words if the related bits in the representation are not in closer positions inside the chromosome.

In this work an Epistasis Reducer Algorithm (ERA) is presented. It is used to preprocess SAT instances for reducing the epistasis of its representation and in this way to improve the performance of a simple genetic algorithm (using classical crossover) when used to solve SAT formulae. The principle of ERA is to transform the original problem by rearranging the variables to satisfy the condition that the most related ones are in closer positions inside the chromosome.

The rest of this paper is organized as follows: In Section 2, a detailed description of the ERA procedure is presented. Section 3 focuses on the simple genetic algorithm used. In section 4, the results of experiments and comparisons are presented and in last section, conclusions are discussed.

2. The ERA method

2.1. Some definitions

The SAT problem can be represented using hypergraphs [17], where each clause is represented by a hyperedge and each variable is represented by a node. Given the following SAT problem in CNF:

$$(\sim A \vee B \vee \sim C) \wedge (A \vee B \vee \sim E) \wedge (A \vee \sim E \vee \sim F) \wedge (\sim B \vee D \vee \sim G)$$

The resulting hypergraph is shown in Figure 1. A weighted graph can be obtained from the hypergraph SAT representation, where the weights represent how many times the variable *i* is related with the variable *j* (see Figure 2 corresponding to the hypergraph in Figure 1). Under this particular representation, the problem of rearranging the variables to satisfy the condition that the most related variables are in closer positions inside the chromosome, is equivalent to solve the bandwidth minimization problem for the graph.

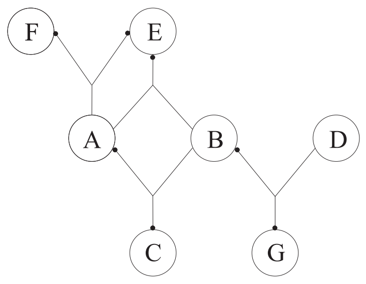

Figure 1. Hypergraph representing a SAT problem.

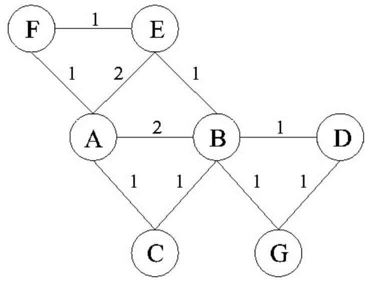

Figure 2. Weighted graph representing a SAT problem.

The bandwidth minimization problem for graphs (BMPG) can be defined as finding a labeling for the vertices of a graph, where the maximum absolute difference between labels of each pair of connected vertices is minimum [3].

Formally, Let $G=(V,E)$ be a finite undirected graph, where V defines the set of vertices (labeled from 1 to N) and E is the set of edges. And a linear layout $\tau=\{\tau_1,\tau_2,...,\tau_N\}$ of G is a permutation over $\{1,2,...N\}$, where τ_i denotes the label of the vertex that originally was identified with the label i. The bandwidth β of G for a layout τ is: $\beta_\tau(G)= Max_{\{u,v\}\in E}|\tau(u)-\tau(v)|$. Then the BMPG can be defined as finding a layout τ for which $\beta_\tau(G)$ is minimum.

2.2. The method

The ERA method is based on a simulated annealing algorithm previously reported in [16], which demonstrated to have competitive results for many classes of graphs. It will approximate the bandwidth of a graph *G* by examining randomly generated layouts τ of *G*. These new layouts are generated by interchanging a pair of distinct labels of the set of vertices *V*. This interchanging operation is called a *move*.

ERA begins initializing some parameters as the temperature, *T*; the maximum number of accepted moves at each temperature, *max_moves*; the maximum number of moves to be attempted at each temperature, *max_attempted_moves*; *max_frozen* is the number of consecutive iterations allowed for which the number of accepted moves is less than *max_moves*; and the cooling rate *cool_rate*. The algorithm continues by randomly generating a move and then calculating the change in the cost function for the new labeling of the graph. If the cost decreases then the move is accepted. Otherwise, it is accepted with probability $P(\Delta C)=e^{-\Delta C/T}$ where *T* is the temperature and ΔC is the increase in cost that would result from a particular move. The next temperature is obtained by using the relation $T_n=T_{n-1}*cool\_rate$. The minimum bandwidth of the labelings generated by the algorithm up that point in time is *min_band*. The number of accepted moves is counted and if it falls below a given limit then the system is *frozen*.

The parameters of the ERA algorithm were chosen taking into account our experience, and some related work reported in [12] and [15]. It is important to remark that the value of *max_moves* depends directly on the number of edges of the graph, because more moves are required for denser graphs; the value of *max_attempted_moves* is set to a large number (*5*max_moves*), because few moves will result in bigger bandwidths. The *max_frozen* parameter that controls the external loop of our algorithm is set to *10*. By modifying these three parameters one can obtain results more quickly, but probably they will not be as close to $\beta(G)$. According to the experiment results the above values give a good balance between the quality of the results and the computational effort required.

3. GA for solving SAT problems

In order to demonstrate the benefits to preprocess SAT instances using ERA, a simple genetic algorithm which uses classical genetic operators was programmed. Next the main implementation details of the GA used in the present work are described.

Chromosome Definition. For the particular case of the SAT problem, given that it consists in a search over *N* boolean variables, this results in a search space of size 2^N, the most natural internal representation is: binary strings of length *N*.

Fitness Function. The choice of the fitness function is an important aspect of any genetic optimization procedure. The fitness function used is the simplest and most intuitive one, the fraction of the clauses that are satisfied by the assignment. More formally: $f(chromosome)=\Sigma f(c_i)/C$, where the contribution of each clause $f(c_i)$ is *1* if the clause is satisfied and *0* otherwise.

Operators. Recombination is done using the two-point crossover operator, and mutation is the standard bit flipping operator. The former has been applied at a 70% rate, and the later at a 0.01% rate. Selection operator is similar to the tournament selection [2], randomly three elements of the population are selected and the one with the best fitness is chosen.

Population Size. In all cases the population size has been held fixed at: *1.6*N*, where *N* is the number of variables in the problem.

Termination Criteria. This GA is terminated when either a solution that satisfies all the clauses in the SAT problem is found, or the maximum number of generations is reached (*300*).

4. Computational results

The ERA method and the GA have been implemented in C programming language and ran into a Pentium 4 1.7 Ghz. with 256 MB of RAM. To test the algorithms described above, several SAT instances as those of the second DIMACS challenge and flat graph coloring instances were used, since they are widely-known and easily available from the SATLIB benchmarks (http://www.satlib.org/benchm.html).

The experimentation methodology used consistently throughout this work was as follows: For each of the selected SAT instances *20* independent runs were executed, *10* using the ERA preprocessing algorithm plus a GA (ERA+GA), and *10* solving the problem solely with the use of the same GA. All the results reported here, are data averaged over the *10* corresponding runs.

Table 1 presents the comparison between ERA+GA and GA. Data included in this comparison are: name of the formula; number of variables N; number of clauses M; β_i and β_f represent the initial and final bandwidth obtained with ERA; the CPU time in seconds used by the ERA algorithm is T_β. Additionally, for each approach it is presented: the number of satisfied clauses S, and the CPU time in seconds T. It is important to remark that times presented for the ERA+GA approach take into account the CPU time used for the preprocessing algorithm.

Table 1. Comparative results between ERA+GA and GA.

			ERA			*ERA+GA*		*GA*	
Formulas	***N***	***M***	β_i	β_f	T_β	***S***	***T***	***S***	***T***
aim100-1_6y★	100	160	94	37	3.1	160	8.05	159	16.81
aim100-2_0y★	100	200	96	42	5.2	200	9.56	198	10.11
dubois28	84	224	77	9	1.0	223	3.08	221	7.25
dubois29	87	232	81	9	1.0	231	2.09	231	19.89
dubois30	90	240	87	9	1.0	239	14.35	237	27.81
dubois50	150	400	144	10	2.1	397	4.03	395	39.94
flat30-1★	90	300	85	26	2.2	300	18.81	299	27.41
flat30-2★	90	300	81	24	2.2	300	13.64	300	17.82
pret60_75	60	160	56	10	1.0	159	2.75	159	14.67
pret150_40	150	400	143	24	2.1	399	35.73	396	23.84
pret150_60	150	400	140	27	3.1	399	15.85	397	21.31
pret150_75	150	400	142	23	3.0	397	38.82	397	65.76

The results of experimentation presented showed that ERA+GA outperforms GA in the selected SAT instances, not only in solution quality but also in computing time. The set of four satisfiable instances, used in the experiments and marked with a star in Table 1, were solved by ERA+GA while only one of them could be solved by GA. Better quality solutions for the ERA+GA is possible thanks to the representation used by the GA ran after ERA, which has smaller epistasis. Smaller computing time for the ERA+GA is possible because the ERA algorithm is able to find good solutions in reasonable time and the GA takes less CPU time.

In Figures 3 and 4 are clearly appreciated the advantages to use the proposed preprocessing algorithm when SAT problems are solved. They show the convergence process of the compared algorithms. The X axis represents CPU time in seconds used by the algorithms, while the Y axis indicates the number of satisfied clauses. It is important to point out that the

ERA+GA curve initiates at the time T_β, which is the CPU time in seconds consumed in the preprocessing stage.

In Figure 5 we explore the behavior of a GA ran on the pret150_75 problem varying the epistasis (controlled by the bandwidth of the associated weighted graph). From this figure it is clear that the performance of the GA is the best when the epistasis is low, i.e. the most related variables are in closer positions inside the chromosome.

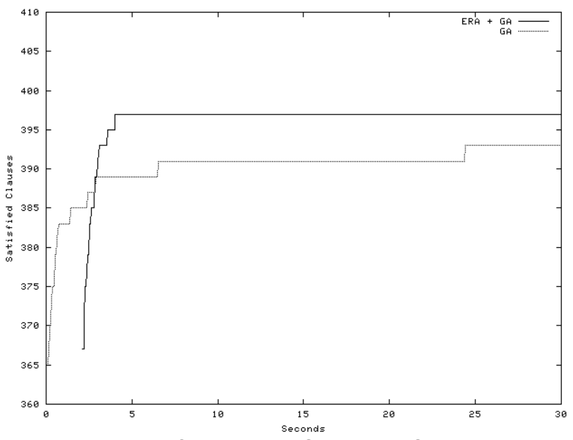

Figure 3. Average curves for ERA+GA and GA on dubois50 problem.

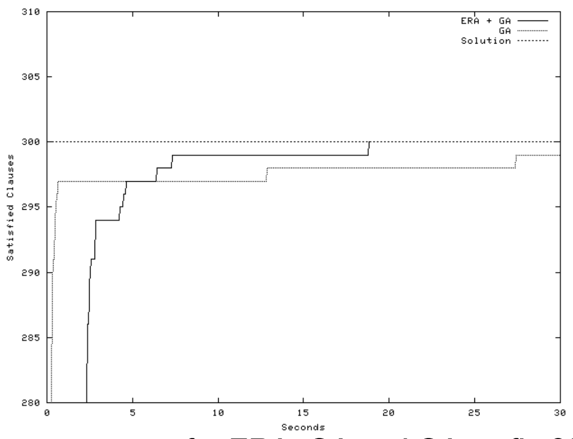

Figure 4. Average curves for ERA+GA and GA on flat30-1 problem.

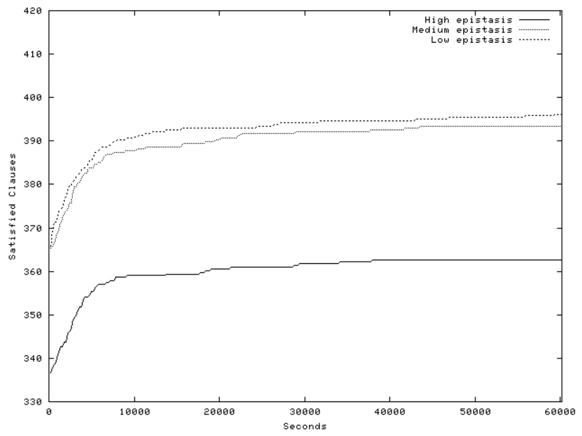

Figure 5. Average curves for a GA ran with different levels of epistasis on pret150_75 problem.

5. Conclusions

In this paper a novel preprocessing algorithm was introduced, called ERA, which improves the performance of GAs when used to solve SAT problems. This approach has been compared versus a simple GA without preprocessing using a set of SAT instances. The ERA+GA outperforms GA in all the selected SAT instances. Also it was evidenced that performance of a GA is improved when the epistasis level is low.

The ERA+GA approach is very promising, taking into account that NP-complete problems can be transformed into an equivalent SAT problem in polynomial time [5], because it opens the possibility to solve through ERA+GA many NP-complete problems which do not have effective representations in GAs.

References

1. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, “Symbolic Model Checking without BDDs”, Proceedings of Tools and Algorithms for the Analysis and Construction of Systems (TACAS’99), Number 1579 in LNCS, Springer Verlag, 1999, pp. 193—207.
2. T. Blickle and L. Thiele. “A mathematical analysis of tournament selection”. In Proceedings of the Sixth ICGA, pages 9—16. Morgan Kaufmann Publishers, San Francisco, Ca., 1995.
3. E. Cutchill and J. McKee, “Reducing the bandwidth of sparse symmetric matrices”, Proceedings 24th National of the ACM (1969), 157—172.
4. Y. Davidor, “Epistasis Variance: A Viewpoint of GA-Hardness”, Proceedings of the Second Foundations of Genetic Algorithms Workshop, Morgan Kaufmann, 1991, pp. 23—35.
5. M. R. Garey and D. S. Johnson, “Computers and intractability: A guide to the theory of NP-completeness”, W.H. Freemanand Company, New York, 1979.
6. E. Giunchiglia, F. Giunchiglia, and A. Tacchella, “SAT-Based Decision Procedures for Classical Modal Logics”, Highlights of Satisfiability Research in the Year 2000, 2000.
7. D. E. Goldberg, “Genetic Algorithms in Search, Optimization and Machine Learning”, Addison-Wesley Publishing Company, Inc., 1989.
8. J. K. Hao, “A Clausal Genetic Representation and its Evolutionary Procedures for Satisfiability Problems”, Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms (France), April 1995.
9. J. K. Hao and R. Dorne. “A new population-based method for satisfiability problems”. In Proceedings of 11th European Conference on Artificial Intelligence (ECAI’94), pages 135—139, Amsterdam, August 1994. John Wiley and Sons, Ltd.
10. J. Holland. “Adaptation in natural and artificial systems”. Ann Arbor: The University of Michigan Press, 1975.
11. H. Kautz and B. Selman, “Planning as Satisfiability”, Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 92), 1992, pp. 359—363.
12. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing”, Science 220 (1983), 671—680.
13. B. Selman, H. Levesque, and D. Mitchell, “A New Method for Solving Hard Satisfiability Problems”, Proceedings of the Tenth National Conference on Artificial Intelligence (San Jose CA), July 1992, pp. 440—446.
14. J. Smith. “On Appropriate Adaptation Levels for the Learning of Gene Linkage”. Journal of Genetic Programming and Evolvable Machines, 3:129—155, 2002.
15. W. M. Spears, “Simulated Annealing for Hard Satisfiability Problems”, Tech. Report AIC-93-015, AI Center, Naval Research Laboratory, Washington, DC 20375, 1993.
16. J. Torres-Jimenez and E. Rodriguez-Tello, “A New Measure for the Bandwidth Minimization Problem”, Proceedings of the IBERAMIA-SBIA 2000, Number 1952 in LNAI (Antibaia SP, Brazil), Springer-Verlag, November 2000, pp. 477—486.
17. I. Vazquez-Moran and J. Torres-Jimenez, “A SAT Instances Construction Based on Hypergraphs”, WSEAS Transactions on Systems 1 (2002), no. 2, 244—247.