ERA: An Algorithm for Reducing the Epistasis of SAT Problems

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 preprocesed 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)[1], is there a truth assignment for the literals that satisfies it?. SAT, is of great importance in computer science both in theory and in practice. In theory SAT is one of the basic core NP-complete problems. In practice, it has become increasingly popular in different research fields, given that several problems can be easily encoded into propositional logic formulae: Planning [10], formal verification [1], and knowledge representation [6]; to mention only some.

[1] A CNF formula F is a conjunction of clauses C, each clause being a disjunction of literals, each literal being either a positive (x_i) or a negative ($\sim x_i$) propositional variable.

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 paper a preprocessing method is proposed, called ERA (Epistasis Reducer Algorithm), to reduce the epistasis of the representation for a SAT problem, 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: Section 2, presents a description of the epistasis problem. In section 3, a detailed description of the ERA procedure is presented. Section 4 focuses on the genetic algorithm used. In section 5, the results of experiments and comparisons are presented and in last section, conclusions are discussed.

2 Tackling Epistasis

In genetics, a gene or gene pair is said to be epistatic to a gene at another locus[2], if it masks the phenotypical expression of the second one. In this way, epistasis expresses links between separate genes in a chromosome. The analogous notion in the context of GAs was introduced by Rawlins [12], who defines minimal epistasis to correspond to the situation where every gene is independent of every other gene, whereas maximal epistasis arises when no proper subset of genes is independent of any other gene. In this case the generation of building blocks will not happen.

The problems of epistasis may be tackled in two ways: as a coding problem, or as a GA theory problem. If treated as a coding problem, the solution is to

[2] The locus is the position within the chromosome.

find a different representation and a decoding method which does not exhibit epistasis. This will then allow conventional GA to be used. In [18] it is shown that in principle any problem can be coded in such a way that it is as simple as the "counting ones task". Similarly, any coding can be made simple for a GA by using appropriately designed crossover and mutation operators. So it is always possible to represent any problem with little or no epistasis. However, for hard problems, the effort involved in devising such a coding could be considerable.

In next section an algorithm called ERA, which is designed to tackle the epistasis of the representation in SAT instances, is presented.

3 The ERA Method

3.1 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.

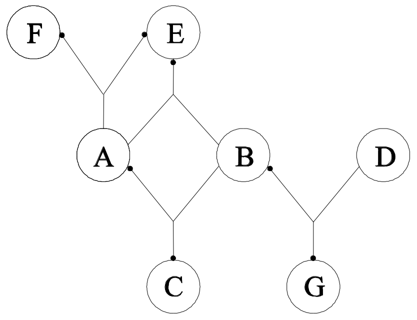

Fig. 1. Hypergraph representing a SAT problem.

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].

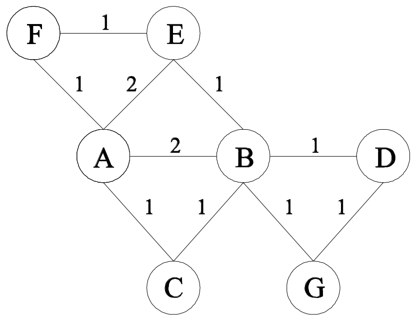

Fig. 2. Weighted graph representing a SAT problem.

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.

3.2 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 labelling 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 labellings 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*.

Next, the ERA procedure is presented:

```
Procedure ERA(G, best_map)
  $T = 0.00004$;   $cool\_rate = 0.85$;
  map = best_map = random labeling;
  sa_band = Bandwidth(G, best_map);
  max_moves = $5 * |E|$;
  max_attempted_moves = 2* max_moves;
  max_frozen = 10;   frozen = 0;
  While (frozen $\leq$ max_frozen)
    moves = attempted_moves = 0;
    While ((moves $\leq$ max_moves) And
            (attempted_moves $\leq$ max_attempted_moves))
      attempted_moves ++;
      a random move is generated, map_ran;
      If (bandwidth decreases Or random_number() $< e^{-\Delta Bandwidth/T}$)
        map = map_ran;   moves ++;
        If (sa_band < Bandwidth(G, map))
          best_map = map;
          sa_band = Bandwidth(G, map);
        End If
      End If
    End While
    T = T * cool_rate;
    If (attempted_moves > max_attempted_moves)
      frozen++;
    Else
      frozen = 0;
    End If
  End While
End ERA
```

The parameters of the ERA algorithm were chosen taking into account our experience, and some related work reported in [11] 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 the experiment results the above values give a good balance between the quality of the results and the computational effort required.

4 GA for Solving SAT Problems

In order to demonstrate the benefits to preprocess SAT instances using ERA a simple genetic algorithm (using classical genetic operators) was implemented. The basic assumption in a GA is the following: a random population of chromosomes (strings of genes that characterize particular solutions) is initially created and exposed to an environment represented by a fitness function; this fitness function is evaluated for each chromosome and only the best-fitted individuals survive and become parents for the next generation. Reproduction is then allowed by splitting the parent chromosomes to create new ones with different genetic information. This procedure is repeated and the population evolves during a given number of generations. Holland [9], introduced the theory of schemas in which he shown that above average-fit schemata get an exponentially increasing number of matches in the population as generations pass.

In the next subsections the main implementation details of the GA used in the present work is described.

4.1 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 in which the *i-th* bit represents the truth value of the *i-th* boolean variable.

This chromosome definition has the following advantages: It is fixed length, binary, context independent in the sense that the meaning of one bit is unaffected by changing the value of other bits, and permits the use of classic genetic operators defined by Holland [9], which have strong mathematical foundations.

4.2 Fitness Function

The choice of the fitness function is an important aspect of any genetic optimization procedure. Firstly, in order to efficiently test each individual and determine if it is able to survive, the fitness function must be as simple as possible. Secondly, it must be sensitive enough to locate promising search regions on the space of solutions. Finally, the fitness function must be consistent: a solution that is better than others must have better fitness value.

In SAT problem the fitness function used is the simplest and most intuitive one, the fraction of the clauses that are satisfied by the assignment. More formally:

$$f(chromosome) = \frac{1}{C}\sum_{c=i}^{c} f(c_i)$$

Where the contribution of each clause $f(c_i)$ is 1 if the clause is satisfied and 0 otherwise.

4.3 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 (these parameters values were adjusted through experimentation).

Selection operator is similar to the tournament selection reported in [2]; randomly three elements of the population are selected and the one with the better fitness is chosen.

4.4 Population Size

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

4.5 Termitation Criteria

The termination condition is used to determine the end of the algorithm. 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 generations).

5 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[3].

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.

Results from ERA algorithm on the selected SAT instances are presented in Table 1. Column titled N indicates the number of nodes in the weighted graph, which represents the original SAT problem. Columns β_i and β_f represent the initial and final bandwidth obtained with the ERA algorithm and the last column presents the CPU time used by ERA measured in seconds.

As can be seen in Table 1, the ERA algorithm provides a balance between solution quality and computational effort. ERA allowed to reduce significantly the bandwidth of the weighted graphs in 5.2 seconds or less, given that the number of vertices was at most 150.

[3] http://www.satlib.org/benchm.html

Formulas	**N**	$\boldsymbol{\beta}_i$	$\boldsymbol{\beta}_f$	$\mathbf{T}_\beta$
aim100-1_6y	100	94	37	3.1
aim100-2_0y	100	96	42	5.2
dubois28	84	77	9	1.0
dubois29	87	81	9	1.0
dubois30	90	87	9	1.0
dubois50	150	144	10	2.1
flat30-1	90	85	26	2.2
flat30-2	90	81	24	2.2
pret60_75	60	56	10	1.0
pret150_40	150	143	24	2.1
pret150_60	150	140	27	3.1
pret150_75	150	142	23	3.0

Table 1. Results from the ERA algorithm.

Table 2 presents the comparison between ERA+GA and GA. Data included in this comparison are: name of the formula; number of variables N; and number of clauses M. Additionally, for each approach it is presented: the number of satisfied clauses S, and the CPU time in seconds. It is important to remark that times presented for the ERA+GA approach take into account also the CPU time used for the preprocessing algorithm.

			ERA+GA		**GA**	
Formulas	**N**	**M**	**S**	**T**	**S**	**T**
aim100-1_6y⋆	100	160	160	8.05	159	16.81
aim100-2_0y⋆	100	200	200	9.56	198	10.11
dubois28	84	224	223	3.08	221	7.25
dubois29	87	232	231	2.09	231	19.89
dubois30	90	240	239	14.35	237	27.81
dubois50	150	400	397	4.03	395	39.94
flat30-1⋆	90	300	300	18.81	299	27.41
flat30-2⋆	90	300	300	13.64	300	17.82
pret60_75	60	160	159	2.75	159	14.67
pret150_40	150	400	399	35.73	396	23.84
pret150_60	150	400	399	15.85	397	21.31
pret150_75	150	400	397	38.82	397	65.76

Table 2. Comparative results between ERA+GA and GA.

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[4] used in the experiments were solved by ERA+GA while only one of them could be solved by GA. Better

[4] In Table 2 a $\star$ indicates a satisfiable formula.

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.

Additionally it has been observed during the experimentation, that as the size of the SAT problem increases, the advantage to use the ERA algorithm also increases. It allows to conclude that it pays to make a preprocessing of SAT problems to reduce the epistasis.

In Figures 3, 4, 5 and 6 is 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 (the results showed are the average of the 10 independent runs). 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.

6 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.

We think that the ERA+GA method is very promising and worthy to more research. In particular it will be interesting and important to identify the classes of SAT instances where is appropriate to apply our method.

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

References

1. Armin Biere, Alessandro Cimatti, Edmund 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*, Proceedings of the Sixth ICGA, Morgan Kaufmann Publishers, San Francisco, Ca., 1995, pp. 9–16.
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 Viewpont 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. Freeman and 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. David 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. Holland, *Adaptation in natural and artificial systems*, Ann Arbor: The University of Michigan Press, 1975.
10. Henry Kautz and Bart Selman, *Planning as Satisfiability*, Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 92), 1992, pp. 359–363.
11. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing*, Science **220** (1983), 671–680.
12. G. J. E. Rawlins, *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, 1991.
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. Jim Smith, *On Appropriate Adaptation Levels for the Learning of Gene Linkage*, Journal of Genetic Programming and Evolvable Machines **3** (2002), 129–155.
15. William M. Spears, *Simulated Annealing for Hard Satisfiability Problems*, Tech. Report AIC-93-015, AI Center, Naval Research Laboratory, Washington, DC 20375, 1993.
16. Jose Torres-Jimenez and Eduardo 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. Isaac Vazquez-Moran and Jose Torres-Jimenez, *A SAT Instances Construction Based on Hypergraphs*, WSEAS Transactions on Systems **1** (2002), no. 2, 244–247.
18. M. Vose and G. Liepins, *Schema Disruption*, Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, 1991, pp. 237–242.

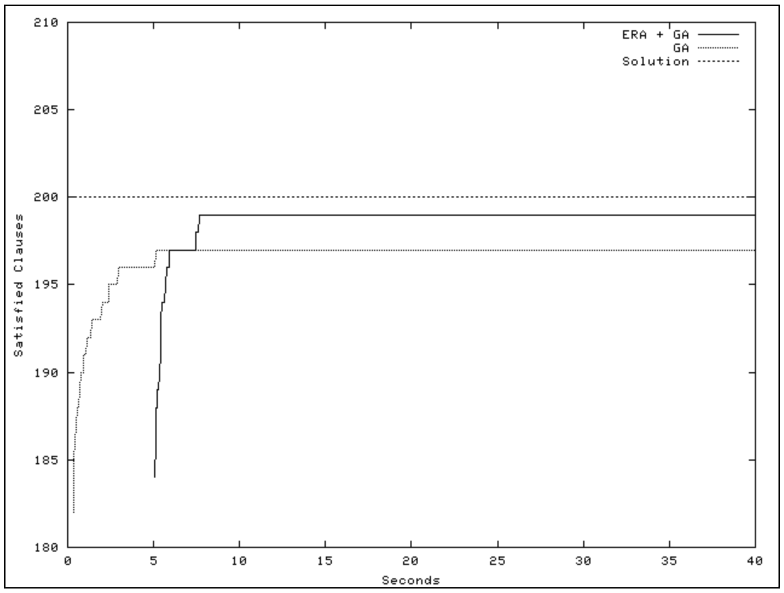

Fig. 3. Average curves for ERA+GA and GA on aim100-2_0y problem.

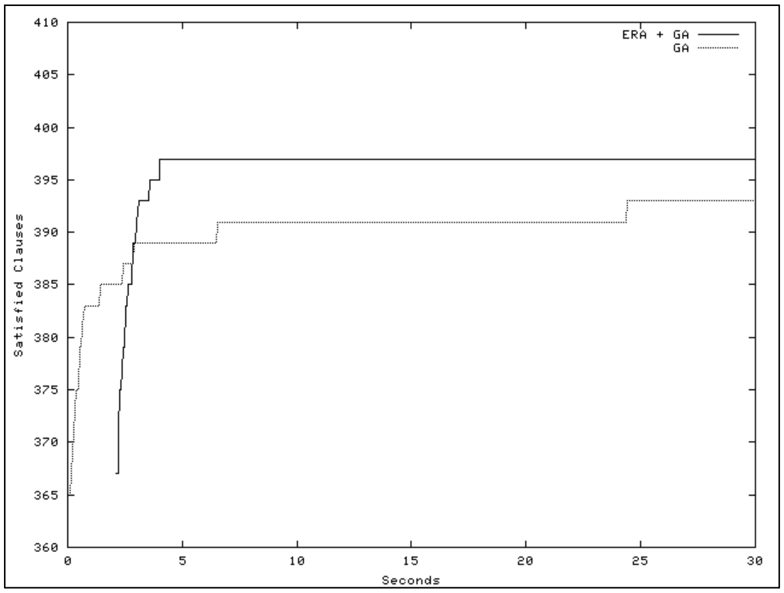

Fig. 4. Average curves for ERA+GA and GA on dubois50 problem.

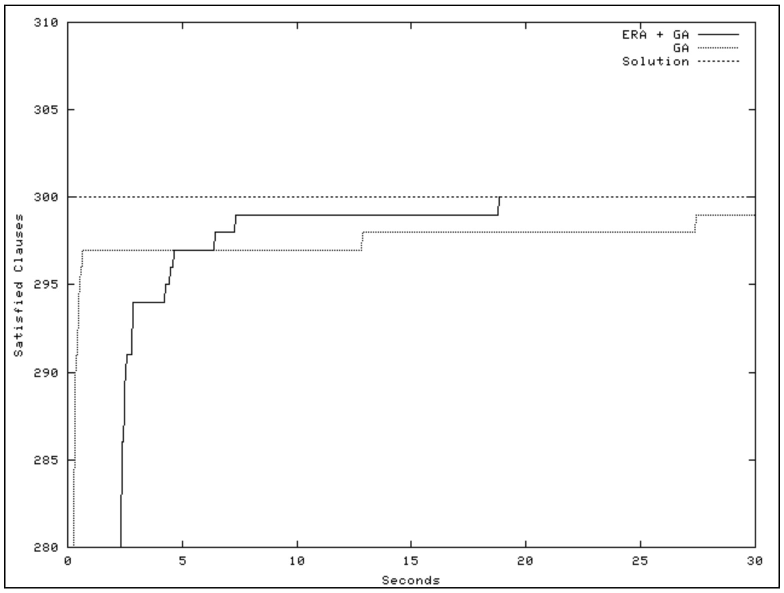

Fig. 5. Average curves for ERA+GA and GA on flat30-1 problem.

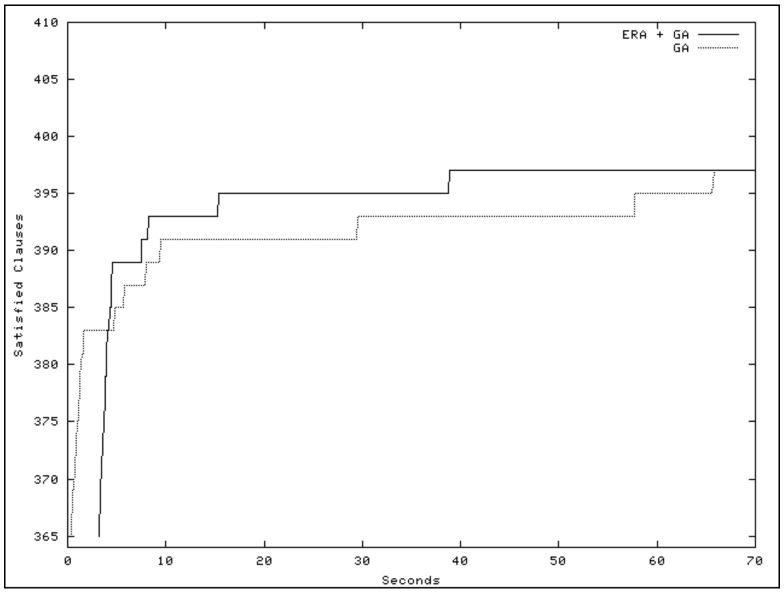

Fig. 6. Average curves for ERA+GA and GA on pret150_75 problem.