A CLAUSAL GENETIC REPRESENTATION AND ITS EVOLUTIONARY PROCEDURES FOR SATISFIABILITY PROBLEMS

Jin-Kao Hao
LGI2P, EMA-EERIE
Parc Scientifique Georges Besse
F-30000 Nîmes, France
email: hao@eerie.fr

Abstract

This paper presents a clausal genetic representation for the satisfiability problem (SAT). This representation, CR for short, aims to conserve the intrinsical relations between variables for a given SAT instance. Based on CR, a set of evolutionary algorithms (EAs) are defined. In particular, a class of hybrid EAs integrating local search into evolutionary operators are detailed. Various fitness functions for measuring clausal individuals are identified and their relative merits analyzed. Some preliminary resluts are reported.

1. Introduction

The satisfiability problem or SAT [3] is of great importance in computer science both in theory and in practice. The statement of the problem is very simple. Given a well-formed boolean formula F in its conjunctive normal form (CNF)[1], is there a truth assignment that satisfies it? In theory, SAT is at the very core of NP-complete problems. In practice, many applications can be formulated with it. One important topic related to SAT is to find such a satisfiable truth assignment for a satisfiable formula F. In this paper, we are interested in this model-finding problem for SAT.

Although SAT is NP-complete, many methods have been devised for practical needs. These methods can be roughly classified into two large categories: complete and incomplete methods. As examples of the first category, we can mention the logic-based approach (Davis & Putnam algorithm, Binary Decision Diagrams), the constraint-network-based approach (CSP, local propagation), and 0/1 linear programming [1]. The second category includes such methods as simulated annealing [14, 15], local and greedy search [6, 9, 13, 14] and evolutionary algorithms (EAs) [2, 16, 7, 3].

EAs are essentially based on three elements: an internal representation, an external evaluation function and an evolution mechanism. It is commonly recognized that the internal representation greatly conditions the success or failure of an evolutionary algorithm. The integration of specific knowledge at different levels in an evolutionary algorithm may also improve much its performance.

SAT has a natural binary representation (what we call variable-coding): an individual represents a potential solution to the boolean formula F, a gene coding a variable of F. Genetic algorithms can be easily developed using the variable-coding and the traditional genetic operator set. This representation is also the one used by all the reported work mentioned above.

Surprisingly, however, no efficient EA has been found yet using this variable-based representation. In fact, the reported EAs for SAT are far from being comparable with GSAT [13, 14], a simple local search based procedure using the same variable-coding. Why this happens is not clear, but we have enough reasons to think that we can devise EAs outperforming GSAT. First, the evolutionary framework allows GSAT to be simulated with a singleton population and a controlled mutation operator. Therefore, we have already a special EA which can do as well as GSAT does. Second, we can integrate easily the basic idea of GSAT into EAs, which should lead to better performance. We are working on these issues, results of that will be reported elsewhere.

Another aspect for designing efficient EAs consists of looking for more intelligent codings, which is the topic of the current paper. The proposed coding is based on the following observation. For a given CNF formula, there are some intrinsical relations among the clauses of the formula. These relations are defined by shared variables and important for problem-solving (model-finding here). However, they are lost in the variable-coding in which only the values of variables are manipulated. In this paper, we explore an alternative coding called clausal representation (CR)

[1]. A CNF formula is a set of clauses linked by logic **and**, a clause being a set of literals linked by logic **or**, a literal being a boolean variable or its negation.

which take into account these relations between clauses.

The paper is organized as follows. In §2, the clausal representation is defined. In the next two sections, various fitness functions and CR-based hybrid EAs are presented. In particular, the integration of local search within the evolution mechanism are detailed. In the last two sections, preliminary experimental results and future work are discussed.

2. CR-A Genetic Clausal Representation

In order to simplify our presentation, without losing the generality, we will consider the 3-SAT problem. Recall that a 3-SAT instance F is defined by a set of L clauses of length 3 connected by logic ***and*** ($\wedge$), L being called the length of F. Each clause is composed of 3 different literals linked by logic ***or*** ($\vee$). A literal is either a boolean variable (called a positive literal) or the negation of a boolean variable (called a negative literal). Note that 3-SAT is the preferred form of many studies, and most methods mentioned above deal with 3-SAT instances.

Given a 3-SAT instance $F \equiv C1 \wedge C2 \wedge ... \wedge CL$ containing N different variables. We encode clauses instead of variables. More precisely, for any clause Ci, there are in total $2^3=8$ possible assignments {000, 001, 010, 011, 100, 101, 110, 111}. Among them, there are exactly 7 satisfying assignments and 1 non-satisfying one. For example, {100} is the only non-satisfying assignment for the clause ($\neg a \vee b \vee c$). Since this non-satisfying assignment will never participate in any solution for SAT instances containing this clause, it will be advantageous to eliminate this assignment from chromosomes at the beginning. This analysis leads us to the following genetic clausal representation (CR).

For any N-variable SAT instance $F \equiv C1 \wedge C2 \wedge ... \wedge CL$, each individual (chromosome) is composed of a string having a length 3*L divided into L parts (genes) <G1, G2,...,GL>, each gene Gi ($1 \le i \le L$) taking as its values a *satisfying* assignment of the corresponding clause Ci. In other words, we will forbidden the non-satisfying assignments to be part of any chromosome. For example, {010 101} is one valid chromosome for the formula $F \equiv (\neg a \vee b \vee c) \wedge (a \vee \neg c \vee b)$ while {100 101} is not since 100 is not a satisfying assignment for the first clause.

One interesting property of CR is that each gene of such a chromosome is locally *consistent* with its clause, which is a necessary condition for any solution[1]. Moreover, the elimination of non-satisfying values from chromosomes implies the reduction of the search space from the beginning. Also the relations between clauses become more explicit in CR and may be efficiently exploited.

With CR, a variable may have the two values (0/1) in a chromosome. Such a variable is said to be *inconsistent*. It is easy to see that a chromosome is a solution iff each of the N variables of the formula F has a unique 0 or 1 value in the chromosome. Consequently, the number of inconsistent variables can be used as a good indicator to measure the fitness of individuals.

3. Fitness Evaluation

The aim of this evaluation is to measure the "fitness" of each individual (chromosome) with respect to the given formula. The difficulty concerning this evaluation is that no exact measuring rule is available and only approximate rules can be tried. Several possibilities exist to define such a fitness function **Eval** with our clausal representation. The basic idea is to use the number of inconsistent variables to define these evaluation functions since we want to reduce this number to 0. In the following, we give some of such functions. Each evaluation function **Eval_i(CH, F)** takes as its input a chromosome **CH** and a CNF formula **F**, and gives an integer or a real number as the fitness of **CH** w.r.t. the given formula **F**.

a) $Eval\_1=\sum W(Vi)$ $(1 \le i \le N)$ where Vi are variables
W(V)=1 if variable V is inconsistent
W(V)=0 otherwise
This function simply counts the number of inconsistent variables. It is easy to see that the value returned by this function is between 0 (which means a solution) and N (which means all variables in the chromosome are inconsistent). The smaller the score, the better the chromosome. However, this function cannot differentiate two chromosomes which have the same number of inconsistent variables.

b) $Eval\_2=\sum C(Vi)$ $(1 \le i \le N)$ where
C(V) = # of clauses containing V or $\neg V$ if variable V is inconsistent
C(V) = 0 otherwise
This function, which is also defined with inconsistent variables, takes into account the importance of variables[2] in order to differentiate chromosomes containing the same number of inconsistent variables.

c) $Eval\_3=\sum |N0(Vi)-N1(Vi)|$ $(1 \le i \le N)$ where

[1]. Note that this is not case with the variable-coding.

[2]. The importance of a variable is defined by the number of its appearances in the formula.

N0(Vi) = # of 0s taken by Vi in CH.
N1(Vi) = # of 1s taken by Vi in CH.

The value returned by this function is an integer between 0 (all variables have the same number of 0s and 1s) and $\sum$(N0(Vi)+N1(Vi)) (all variables have an unique value 0 or 1). A CH having this biggest value is a solution while a CH having the value 0 can be considered to be less fitted.

d) Eval_4=$\sum|1-2*N1(Vi)/N01(Vi)|$ (1^2i^2N) where
N1(Vi) = # of 1s taken by Vi in CH.
N01(Vi) = # of 0s and 1s taken by Vi in CH, which is the total number of literals of Vi.

This function returns a real number between 0 and N. The value 0 means that all the variables have the same number of 0s as 1s while the value N corresponds to a solution.

Moreover, it is possible to combine some of them to define multiple-step evaluation. Finally, instead of defining these functions statically, they may be made self-adapted during the evolution of the population.

4. CR-based Evolutionary Procedures

Using the clausal representation, various evolutionary algorithms can be directly built with a population of clausal chromosomes. Operators such as random crossover and mutation can be directly applied to this population with some minor constraints. For instance, in order for crossover to be meaningful, a single point crossover will take place at a point 3*i ($1<i<L-1$, L being the length of the formula F). Mutation will change a 3-bit value of a selected gene to another 3-bit valid value. We will not detail this kind of algorithm since it is straightforward to build them. In the following, we will first investigate a special population for clausal individuals. Then we investigate a class of hybrid algorithms based on this population.

In order to build an evolutionary procedure, an initial population must be generated in some way. Although this population may contain any number of chromosomes, a more intelligent organization can be devised with the clausal representation. In fact, since each gene in CR corresponds to a clause which has exactly 7 satisfying assignments, one natural way of realizing the initialization is to generate a population of 7 chromosomes such that for each gene we make sure that the gene has all of its 7 satisfying assignments (alleles). Here is an example.

$$F \equiv (\neg a \vee b \vee c) \wedge (a \vee \neg c \vee b) \wedge (c \vee \neg a \vee \neg b) \wedge (\neg a \vee \neg c \vee \neg b)$$

Population

Clauses	C1	C2	C3	C4
Genes	G1	G2	G3	G4
Ind.1	0 0 0	0 0 0	0 0 0	0 0 0
Ind.2	0 0 1	**0 0 1**	**0 0 1**	**0 0 1**
Ind.3	**0 1 0**	0 1 1	0 1 0	0 1 0
Ind.4	0 1 1	1 0 0	1 0 0	0 1 1
Ind.5	1 0 1	1 0 1	1 0 1	1 0 0
Ind.6	1 1 0	1 1 0	1 1 0	1 0 1
Ind.7	1 1 1	1 1 1	1 1 1	1 1 0
Forbidden-values	1 0 0	0 1 0	0 1 1	1 1 1

The 7 possible values for each gene are put in increasing order in this example; the order is not important in the general case.

One nice property of this population P is that all the solutions for a given formula are covered by it. In fact, any solution may be "constructed" from P by concatenating good values of the genes. This is because a solution will be necessarily a combination of all the L genes, each taking a value from 7 satisfying ones for a clause. Therefore, what an EA must do is to make these chromosomes evolve towards solutions by using some evolution mechanisms. In the following, we will see how hybrid EAs can be built using such a population.

There are many good reasons and different ways to build hybrid EAs. We will not discuss this general issue here, see for example [10] for more detail. We will instead consider the hybridization of the evolutionary framework with local search using our clausal coding.

Local search groups a class of heuristics based on random search and neighborhood relations. They are known to be very efficient for finding local optima for certain classes of problems. A typical local search procedure begins with a starting point *s0*, often generated randomly. Then an iterative process follows to produce a series of points *s1*,*s2*,...,*sn*. The last point *sn* will be the best solution found. Each *si* (1^2i^2n) is chosen from a set of neighbors associated to *si-1* which are defined by a neighborhood function. In essence, a local search procedure is characterized by this neighborhood function and the neighbor selection strategy (that decide which the neighbor to take). Various hillclimbers (steepest ascent, next ascent), simulated annealing [7] and Tabu search [4] are typical examples of local search. Integrating local search into EAs makes it possible to combine the diversity offered by EAs and the efficiency of local search.

There are essentially two possibilities to realize this hybridization. First, we can use the local search to improve all the individuals or a subset of a population at each generation. This integration which is easy to realize can be qualified loosely coupled approach.

Second, local search can be directly incorporated into genetic operators. More pricisely, applying local search at the level of genetic operators consists in using *controlled* crossover or mutation to improve locally the quality of some chromosomes. For example, crossover can produce, according to the local search techniques used, the best children or the next better ones. Similarly, mutation can look for the best or next better offspring. If a local search technique like simulated annealing is used, degenerate offsprings might also be accepted with certain fixed or variable probability.

The following pseudo code gives an example in which we decide to improve a single (the first) chromosome by local search. For convention, we use P[I,J] to indicate the Jth gene of the Ith chromosome ($1 \le I \le 7$, 7 being the population size, $1 \le J \le L$, L being the length of the SAT instance F), and we use CH[I] to represent the Ith chromosome of P.

```
Procedure Local_Genetic
{
1. generate (P);                    /* as described above */
2. while not stop_condition do
3. for J=1 to L                     /* L=length of F */
4.    for I=2 to population_size    /* equal to 7 */
5.       if improve(P[1,J], P[I,J])
6.       then swap(P[1,J], P[I,J]);
}
```

The "improve" operation tests if swaping the two alleles for a given gene improves the fitness of the given chromosome. This test can be carried out by using the evaluation functions described in §3. Remember that an chromosome is a solution iff all its variables have a unique 0/1 value. Thus improving an individual means to reduce the number of its inconsistent variables. The "swap" operation takes the corresponding genes of two chromosomes and exchanges effectively their alleles. It is interesting to note that "swap" can be considered as either mutation or two-point crossover. Note also that selection is realized implicitly; the two children produced by "swap" replace their parents. The "stop_condition" in the **while** loop may be defined according to different criteria. One possible way is to fix the number of generations. Another way is to check if the chromosome which is being locally improved is effectively improved during the last iteration. The complexity analysis of this algorithm is straightforward.

From this simple algorithm, many variants can be easily deduced. Here are some examples, First, this algorithm tries systematically to improve all of the L genes of a given chromosome. A test can be added between the lines 3 and 4 to select only the genes containing at least an inconsistent variable. Second, for a given gene, the algorithm may swap at most 6 alleles due to the **for** loop at the line 4. This loop can be replaced by a procedure which may look for the next better allele or the best one, and then swap only once. Third, this algorithm look for improving only a single chromosome. It can be easily modified to allow all the chromosomes of the population to be improved at each **while** loop. Finally, classic crossover can be similarly adapted to include local search.

5. Experimental Results

Experimentations are being carried out. To test the different methods described above, we are using such SAT instances as those of the second Dimacs challenge and random instances as described in [10]. We observe that the methods based on local search are very efficient to reach local optima (which may or may not be solutions). Very often, local optima can be found after a small number of generations for formulae containing 50, 100, 200 variables. However, it is too early to make any final conclusion yet given the fact that only partial experiments have been realized. We hope more results will be available in the near future.

6. Discussion & Conclusions

We presented an original genetic representation for SAT based on clause assignments and its associated evolutionary procedures. In particular, a class of procedures integrating the local search within the evolution mechanism are introduced. Several evaluation functions are identified.

Experiments are being carried out to study the effectiveness of these ideas. The limited experimental results prevent us to give any conclusive remarks, but show that these procedures need only a small number of generations to find (local or global) optima for the set of SAT instances tested. Note that the population size (only 7 individuals) used is dramatically small compared with the search space visited.

One possible disadvantage of the clausal-coding may be its bigger search space compared with that of the variable-coding given the fact that there are often more clauses than variables in hard SAT instances. Also there are more alleles for each gene with the clausal-coding than with the variable-coding. However, with the clausal coding which conserves relations among variables, there may exist more room for us to devise more accurate fitness functions and some intelligent genetic operators which permit to exploit efficiently these relations.

Another interesting point worth of studing is the landscapes related to the two different codings. For example, the distributions and depths of local optima of these two codings are *a priori* very different. A better understanding of these issues will help to design more appropriate fitness functions and genetic operators.

We continue to experiment various EAs for SAT using both the variable-coding and clausal-coding. We wish to be able to compare the two codings and their related algorithms. Finally, the ultimate goal of our study on SAT is to develop hybrid EAs (which may use either of the two codings) more efficient than the simple local search procedures like GSAT.

Acknowledgments

The author would like to thank R. Dorne and S. Vautier for the many useful discussions. The work is partially supported by the Bahia project (PRC-IA the French nationl coordinated research programme for artificial intelligence).

References

[1] BAHIA project. "Comparative study of three formalisms in propositional logic" (in French). 5ème Journées Nationales PRC-GDR en Intelligence Artificielle, Nancy, February 1995.

[2] De Jong K.A. & Spears W.M. "Using genetic algorithms to solve NP-complete problems". Intl. Conf. on Genetic Algorithms (ICGA'89), Fairfax, Virginia, June 1989, pp124-132.

[3] Dorne R. & Hao J.K. "New genetic operators based on a stratified population for SAT" (in French). Evolution Artificielle, Toulouse, France, September 1994.

[4] Garey M.R. & Johnson D.S. "Computers and intractability: a guide to the theory of NP-completeness". Freeman, San Francisco, CA, 1979.

[5] Glover F. & Laguna M. Tabu search. in [12].

[6] Gu J. "Efficient local search for very large-scale satisfiability problems". SIGART Bulletin, Vol. 3, No. 1, January 1992.

[7] Hao J.K. & Dorne R. "A new population-based method for satisfiability problems". Proc. of 11th European Conf. on Artificial Intelligence (ECAI'94), Amsterdam, Aug. 94, pp135-139.

[8] Kirkpatric S., Gelatt C.D. & Vecchi M.P. "Optimistion by Simulated Annealing". Science, Vol. 220, May 1983, pp671-680.

[9] Koutsoupias E. & Papadimitriou C.H. "On the greedy algorithm for satisfiability". Information Processing Letters, Vol. 43 1992, pp53-55.

[10] Lawrence D. Handbook of genetic algorithms, Van Nostrand Reinhold, New York, 1991.

[11] Mitchell M., Selman B. & Levesque H.. "Hard and easy distributions of SAT problem". Proc. of the AAAI '92, an Jose, CA, 1992, pp.459-465.

[12] Reeves C.R. (Ed.) "Modern Heuristic Techniques for Combinatorial Problems". Blackwell Scientifi-que Publications, Oxford, 1993.

[13] Selman B., Levesque H. & Mitchell M. "A new method for solving hard satisfiability problems". Proc. of the AAAI '92, San Jose, CA, 1992, pp.440-446.

[14] Selman B., Kautz H.A. & Bram C. "Noise strategies for improving locla search". Proc. of the AAAI '94, Seattle, WA, 1994,.

[15] Spears W.M. "Simulated annealing for hard satisfiability problems". NCARAI Technical Report #AIC-93-015, Washington D.C. July 1993.

[16] Young R.A. and Reel A. "A hybrid genetic algorithm for a logic problem", Proc. of the 9th European Conf. on Artificial Intelligence, Stockholm, Sweden, Aug. 1990, pp.744-746.