Solving the Satisfiability Problem by a Parallel Cellular Genetic Algorithm

Gianluigi Folino, Clara Pizzuti and Giandomenico Spezzano
ISI-CNR c/o DEIS
Università della Calabria
87036 Rende (CS), Italy
email:{folino, pizzuti, spezzano}@si.deis.unical.it

Abstract

This paper presents a new evolutionary method for solving the satisfiability problem. It is based on a parallel cellular genetic algorithm which performs global search on a random initial population of individuals and local selective generation of new strings according to new defined genetic operators. The algorithm adopts a diffusion model of information among chromosomes by realizing a two-dimensional cellular automaton. Global search is then specialized in local search by changing the assignment of a variable that leads to the greatest decrease in the total number of unsatisfied clauses. A parallel implementation of the algorithm has been realized on a CS-2 parallel machine.

1. Introduction

The *satisfiability problem (SAT)* is a fundamental problem in Artificial Intelligence applications, automated reasoning, mathematical logic and related fields. Many practical problems, including block-world planning problems [9], Boolean circuit synthesis problems [8], circuit diagnosis [10], can be formulated as SAT problems. In automated reasoning satisfiability problem plays a key role because of its correspondence with deductive inferencing [2]. In computing theory SAT is the core problem of the family of computationally intractable NP-complete problems [4]. Many of such problems have been identified as central in computing theory and engineering and may be efficiently transformed to SAT.

Formally the satisfiability problem can be formulated as follows. Let $U = \{u_1, \ldots, u_n\}$ be a set of n boolean variables. A truth assignment for U is a function $t : U \rightarrow \{true, false\}$. Corresponding to each variable u are two literals u and $\neg u$. A literal u (resp. $\neg u$) is true under t iff $t(u) = true$ (resp. $t(\neg u) = false$). A set C of literals is called a clause and represents the disjunction (or logical connective) of these literals. A set of clauses is called a formula. A formula f is interpreted as a formula of the propositional calculus in conjunctive normal form (CNF), so that a truth assignment t satisfies a clause C iff at least one literal $u \in C$ is true under t and t satisfies f iff it satisfies every clause in f . For example, let $U = \{u_1, u_2\}$ and $f = \{\{u_1, \neg u_2\}, \{u_1, u_2\}\}$, then $t(u_1) = t(u_2) = true$ satisfies f. The SAT problem consists of a set of n variables $\{u_1, \ldots, u_n\}$, and a set of m clauses $C_1, \ldots, C_m$. The goal of the satisfiability problem is to determine whether there exists an assignment t of truth values to variables that makes the formula $f = C_1 \wedge \ldots \wedge C_m$ in conjunctive normal form satisfiable.

Because of its importance, there has been a lot of interest in developing efficient methods to solve SAT problems. A raw classification of these methods subdivides them into complete and incomplete ones. A method is complete if it is always able to determine whether a formula is satisfiable or unsatisfiable. A method is incomplete if it does not find a solution to SAT even if it exists. In this case the algorithm stops without having found a satisfiable assignment but it is not known if such an assignment does exist. The main drawback of complete methods is that they are computationally heavy when the input size increases. For example, the Davis-Putnam algorithm, one of the fastest developed, performs very poorly with more than 400 variables.

More recently, local search algorithms [5, 15, 16] have received a lot of attention because they have been successfully applied to certain hard classes of large satisfiability problems and they have been shown to out-perform the best known complete methods. Although *SAT* is intractable in the worst case, many instances of the problem are easily solved in practice [3]. Thus the class of random *k-SAT* problems has been defined by generating instances with respect to three parameters: the number n of variables, the number m of clauses and the length k of each clause. Mitchel et al. [12] have shown that with $k = 3$ these problems are very hard when the generated instances are equally likely to be either satisfiable or unsatisfiable, i.e. they are neither underconstrained nor overconstrained. The crossover point occurs when m/n is equal to about 4.23.

Local search is a very efficient technique devised to solve *NP-hard* combinatorial optimization problems. Given an initial point, a local minimum is found by searching for a local neighborhood which improves the value of the object function. The *SAT* problem can be formulated as an optimization problem in which the goal is to minimize the number of unsatisfied clauses. Thus the optimum is obtained when the value of the object function equals zero, which means that all clauses are satisfied.

One of the most popular method for solving *SAT* is *GSAT* [15]. This algorithm starts with a randomly generated truth assignment. It then changes (*flips*) the assignment of the variable that leads to the largest decrease in the total number of unsatisfied clauses. Such flips are repeated until either a satisfying assignment is found or a preset of maximum number of flips is reached. This process is repeated as needed up to a maximum of Max-Tries times. The main problem in applying local search methods to combinatorial problems is that the search space presents a lot of local optima and, consequently, the algorithm can get trapped at local minima. In order to overcome this problem some heuristics (including simulated annealing, random noise [16]) have been implemented. They are based on allowing to move to a new neighborhood point of the local search space even if the value of the object function decreases.

In this paper a new evolutionary method for the satisfiability problem based on a *parallel cellular genetic algorithm (PCGA)* is presented. Global search on a random initial population of individuals is performed by selective generation of strings according to new suitably defined genetic operators. The algorithm adopts a diffusion model of information among chromosomes by realizing a two-dimensional cellular automaton. The neighborhood relation considered, that is the set of cells that neighbor a given cell (i, j), is *Moore neighborhood (8-neighbour)*. Such a model allows for the formation of subpopulations of strings having common characteristics inside the niches and relatively noncompetitive among them. Subpopulations diffuse information slowly thus avoiding to get trapped into local minima too early. Global search is then specialized in local search by adopting the same greedy strategy of GSAT, that is by changing the assignment of a variable that leads to the greatest decrease in the total number of unsatisfied clauses. Because of the underlying cellular framework, a parallel implementation of the algorithm has been realized on a *CS*-2 parallel machine. It is worth note that a classical genetic algorithm (*GA*) has been used by [7] to solve *SAT* problems. However, because of the poor results, the use of parallelism to speed up the execution time and improve convergence is suggested.

The paper is organized as follows. Section 2 contains a brief description of standard Genetic Algorithm (GA) and a presentation of the cellular automata model to enable a fine-grained parallel implementation of GA through the diffusion model. Section 3 describes the parallel cellular genetic algorithm and its genetic operators. Section 4, finally, reports on initial SAT experiments.

2. Cellular genetic algorithms

GA are a class of adaptive general-purpose search techniques inspired by natural evolution. They have been applied to many problems in diverse research and application areas such as hard function and combinatorial optimization, neural nets evolution, planning and scheduling, machine learning and pattern recognition. A standard GA works on a population of elements (*chromosomes*), generally encoded as bit strings, representing a randomly chosen sample of candidate solutions to a given problem. A GA is an iterative procedure that maintains a constant-size population of individuals. Each member of the population is evaluated with respect to a fitness function. At each step a new population is generated by selecting the members that have the best fitness. In order to explore all the search space the algorithm alters the selected elements by means of genetic operators to form new elements to be evaluated. The most common genetic operators are crossover and mutation. Crossover combines two strings to produce new strings with bits from both, thereby producing new search points. Through crossover the search is biased towards promising regions of the search space. Mutation flips string bits at random if a probability test is passed; this ensures that, theoretically, every portion of the search space is explored. GA are stochastic iterative algorithms without converge guarantee. Termination may be triggered by reaching a maximum number of generations or by finding an acceptable solution. GA are often applied in situations where almost no domain knowledge is available, although domain knowledge can be incorporated in the GA by either modifying the genetic operators, choosing a particular initial population, or modifying the quality function.

A drawback of the GA is represented by the number of evaluations required to obtain the solution of a problem. In general, larger populations allow to find the solution in fewer generations because of the larger diversity of configurations of individuals in the populations. However, the evaluation of larger populations may require a huge amount of time. A common approach to reducing this execution time is to resort to parallel GA. Parallel implementation of GA involves two main approaches: the *island* model [11] and the *diffusion* model [13]. The island model divides the population in subgroups, and a sequential GA works on each partition. Infrequently, solutions migrate randomly between subgroups, exchanging genetic information. In the diffusion model each chromosome has a spatial location and interacts only within a particular neighborhood. Information slowly diffuses across the grid thus clusters of solutions are formed around different optima.

Cellular automata (*CA*) [18] can be used as a framework to enable a fine-grained parallel implementation of GA through the diffusion model. A CA is composed of a set of cells in a regular spatial lattice, either one-dimensional or multidimensional. Each cell can have a finite number of states. The states of all cells are updated synchronously according to a local rule, called a transition function. That is, the state of a cell at a given time depends only on its own state at the previous time step and the states of its *nearby* neighbors (however defined) at that previous step. Thus the state of the entire automaton advances in discrete time steps. The global behaviour of the system is determined by the evolution of the states of all the cells as a result of multiple interactions. A cellular genetic algorithm [14] can be designed associating to each cell of a CA a transition function state with two substates: one contains a chromosome belonging to the initial population and the other its fitness. At the beginning a random chromosome is generated and its fitness is evaluated. After, at each generation, the transition function associated with a cell selects the chromosome with the best fitness in the neighborhood. Then the genetic operator of crossover is applied to the current string and the selected string. After evaluating the offspring, if one of them has a better fitness than the current string, it becomes the current string. Next, the mutation operator, with probability *pmut*, is applied to this string.

The cellular genetic algorithm on a 2-dimensional toroidal grid, using the von Neumann neighborhood, can be described by the pseudo-code shown in figure 1.

We use the CARPET language [17] to describe the transition function of each cell of the CA. CARPET is a high-level language based on C language with additional constructs to describe cellular algorithms. The definition part *cadef* sets the features of the CA. In this part the user can define the *dimension*, the *radius*, the *state* and the *neighborhood* of the CA. The *parameter* statement assigns a global value for all the cells of the automaton. This value can be interactively changed by the user interface.

Cadef part in figure 1 defines a two-dimensional CA, with radius equal to 1 and a state with two substates that can be individually accessed. The parameter statement assigns an initial value to the probabilities that are used to execute the crossover and the mutation. The predefined variable cell refers the current cell in the two-dimensional space under consideration. The different substates are referred by appending the name of the substates to the reserved word cell or to variables defined in the neighbor statement. *Neum* is an alias to refer a neighbor cell variable. *Step* is a predefined variable and allows to know the number of iterations that have been executed. The statement *update* updates the value of one of the substates of a cell. After an update execution the value of the substate, in the current iteration, is unchanged. The new value takes effect at the beginning of the next iteration.

This approach has the advantage of working with large populations, enabling fast convergence, and reducing the number of iterations and the execution time. This cellular model avoids the problem of premature convergence in some GA applications, i.e. that a rather good individual, with a fitness higher than the others, spreads rapidly through the population. In cellular implementation of GA the produced information flows and spreads like a slow migration to the zones near the interested neighbor. So good schemata discovered can slowly diffuse through the whole population, leaving time to discover other schemata at different portions. Furthermore, this approach allows to keep a diversity of the population as the search proceeds in the search space.

3. The PCGA method

Analogously to traditional evolutionary approaches, the PCGA method is based on a population of binary strings of length n. Each string encodes an assignment of truth values to the set of n variables in which the

```
cadef
{
  dimension 2 ;
  radius 1 ;
  state ( float chromosome, int fitness);
  neighbor Neum[4] ([0,-1]North,[-1,0]West,
                   [0,1]South, [1,0]East);
  parameter (pcross 0.5, pmut 0.8) ;
}
 float chrom,off1,off2,f1,f2,index,fit,cross,mut ;
 int partner;
{
 if (step = 0)
 {
   chrom = rand_float();
   fit = evaluate_fitness(chrom);
   update (cell_chromosome, chrom) ;
   update (cell_fitness, fit) ;
 }
 else
 {
 while (not solution)
  {
  chrom = cell_chromosome;
  partner = max_fitness(North_fitness,West_fitness,
                 East_fitness, South_fitness);
   cross = rand_float();
   if (cross > pcross)
    {
    crossover(chrom,Neum[partner]_chromosome,
             off1,off2);
    f1 = evaluate_fitness(off1) ;
    f2 = evaluate_fitness(off2) ;
    index = index_max_fitness(cell_fitness,f1,f2) ;
    chrom = find_chrom(index, cell_chromosome,
                  off1, off2);
    }
    mut = rand_float();
    if (mut > pmut)
      mutation(chrom) ;
    fit = evaluate_fitness(chrom) ;
    update (cell_chromosome,chrom) ;
    update (cell_fitness, fit) ;
   }
  }
 }
```

Figure 1. Pseudo-code of the cellular genetic algorithm.

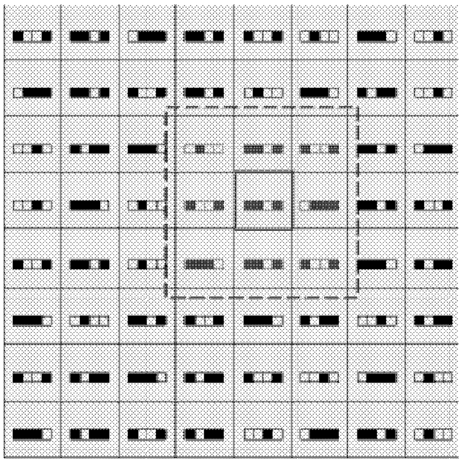

Figure 2. A population with Moore's neighborhood

i-th bit represents the truth value (*true* if the bit value is 1, *false* otherwise) of the *i-th* boolean variable. The fitness function evaluates a string with respect to the number of unsatisfied clauses. A string whose fitness value is zero is a solution to the satisfiability problem because it means that there are no unsatisfied clauses.

As already observed, the search space of combinatorial problems presents a high number of local optima. The implementation of standard genetic operators thus creates some problems. In fact, classical crossover and mutation and the selection mechanism based on the allocation of offspring strings proportionally to the fitness value of their parents, brings the population to a rapid uniformity which corresponds to get trapped into local minima. It has been experimented [6, 7] that it is essential for the success of the method to maintain a good diversity in the individuals which constitute the population because this gives more chance to explore different portions of the search space.

This goal has been obtained by using of a cellular genetic algorithm. The population of strings is mapped into a two-dimensional square lattice, thus every string s represents a cell (i,j) of a two-dimensional cellular automaton. Such a mapping allows to define a neighborhood (that is the set of cells that neighbor s) in which the cell (i,j) can interact. Every string s in the population is thus mated with the element, among the k neighbors, where k depends on the neighborhood relation chosen, with the best fitness.

The standard Moore neighborhood has been experimented. Figure 2 shows a population with such a neighborhood. Each cell contains a transition function like that defined in figure 1.

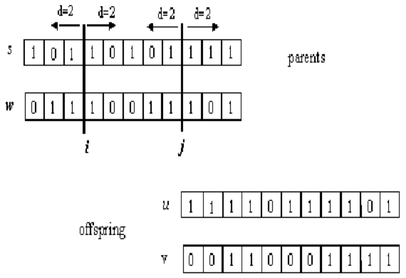

Figure 3. 2-points crossover operator.

The new crossover and mutation operators are defined as follows. Crossover is realized by selecting two positions i and j at random, between 1 and n, and two new strings, u and v, are generated by swapping all the bits of the two parents, s and w, contained into a neighborhood of i and j. The length d of the neighborhood is a parameter of the method (see figure 3).

If the fitness of the offspring is better than that of the current string s, s will be substituted by either u or v, (the one having the best fitness). Such a model allows for the formation of subpopulations of strings having common characteristics inside the niches and relatively noncompetitive among them. Notice that the other parent string w is used only to generate the offspring. Such a choice is very important because subpopulations diffuse information very slowly thus avoiding to bring the population towards an homogeneousness too high which would mean to get trapped into local minima.

The mutation operator plays an important role in the diversity maintenance because it introduces uphill moves that could increase the number of unsatisfied clauses but which, at the same time, employs a mechanism to escape from local minima by allowing to generate strings belonging to completely different portions of the search space.

Bit mutation is done according to the random walk strategy of [16]. The variable with the best decrease in the number of unsatisfied clauses is flipped and, when no downward move is possible, a variable that appears in some unsatisfied clause is picked at random and its assignment value changed.

Thus, whether crossover brings the population towards the best individuals, mutation has both the roles of improving the population and, at the same time, to introduce new strings to maintain a good diversity in the individuals which constitute the population. Furthermore, since the execution of both crossover and mutation could give rise to a solution, the satisfiability test is done after the application of each of them. The combination of the new local selection mechanism along with specialized crossover and mutation operators allows the genetic algorithm to reach a solution in a low number of generations.

4. Implementation and results

The PCGA method is naturally suitable for running on distributed-memory MIMD machines. It uses a grid of elements (population) of fixed size where each cell performs a transition function on a string chosen randomly. PCGA leads the search on a population that can be regarded as a region of the feasible space. Genetic operators generate, at each iteration, a different population and this allows to explore the whole space of the solutions. A larger population allows to explore a wider region of the search space, reduces the number of iterations needed by the algorithm and improves the convergence. Therefore, a parallel implementation of the algorithm, by reducing the execution time of a single iteration, allows to use a larger population to find a better solution in fewer iterations.

An efficient parallel implementation of PCGA can be realized using the SPMD (*Single-Program Multiple-Data*) model. According to this model, the algorithm can be implemented as a set of medium-grain cooperating processes, each mapped on a distinct processing element that executes the same program on different data (a partition of the population).

To implement the PCGA algorithm, we used the parallel computation environment provided by the CAMEL system [1]. CAMEL is an interactive parallel environment allows to develop and execute CARPET programs on a parallel machine. The main goal of CAMEL is to integrate computation, visualization and control into one environment that allows interactive steering of applications. CAMEL consists of

- a parallel run-time support for the CARPET language;
- a graphical user interface (GUI) for editing, compiling configuring, executing and steering the computation;
- a tool for the interactive visualization of the results.

The run-time support is implemented as a SPMD program using the C language plus the standard MPI

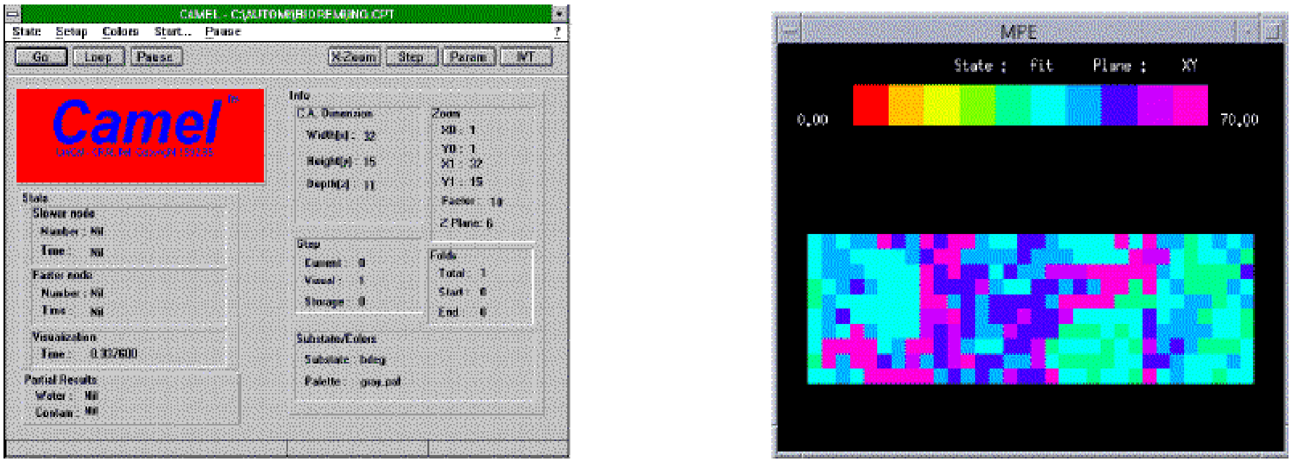

Figure 4. A snapshot of PCGA iteration.

library and can be executed on different parallel machines such as the Meiko CS-2, IBM SP2 and networks of workstations. The concurrent program which implements the architecture of the system is composed by a set of *macrocell* processes, a *controller* process and a *GUI* process. Each macrocell process runs on a single processing element of the parallel machine and contains a strip of cells of the CA. The synchronization of the automaton and the excution of the commands, provided by the user throught the GUI interface, are carried out by the controller process. MPI primitives handles all the communications among the processes using two different *communicators*. A communicator provides a mechanism for distinguishing between messages used for different purposes. CAMEL uses two communicators. One includes the controller and macrocell processes and the other one the controller and the GUI process. In order to improve the performance of the applications that have a diffusive behaviour, CAMEL integrates the run-time support with a load balancing strategy. This load balancing is a domain decomposition strategy similar to the scattered decomposition technique.

The PCGA algorithm is coded in CARPET. CAMEL divides the population in strip, according to the indications provided by the user through the GUI, and assigns each strip to a macrocell process for the execution. The macrocell processes are automatically mapped on the nodes of the parallel machine according to a ring topology.

By the GUI of CAMEL, shown in figure 4, the user can view the evolution of the algorithm by visualizing the output according to the visualization step defined and to change, during the execution time, the parameters that define the length of the neighborhood for the 2-points crossover and the probability with which to perform crossover and mutation.

PCGA starts with a random population and the parameters are fixed for all the computation. Figure 4, shows as our algorithm induces niches formation allowing information to diffuse across the grid and promoting the making of clusters of solution around different optima.

The parallel implementation has been executed on a Meiko CS-2 parallel machine. The CS-2 is a distributed memory MIMD parallel computer. It consists of Sparc based processing nodes running the Solaris operating system on each node, so it resembles a cluster of workstations connected by a fast network. Each computing node is composed of one or more Sparc processors, a communication co-processor, the Elan processor, that connects each node to a fat tree network built from Meiko 8x8 crosspoint switches. Our machine is a 12 processors CS-2 based on 200 MHz HyperSparc processors running Solaris 2.5.1.

PCGA has been tested on hard randomly generated 3-*SAT* problems. In particular, tests with 64 up to 512 variables have been considered. In our experiments, we used a population size of 320 and 2400 elements, a radius equal to 1, a probability between 0.3 and 0.5 for crossover and between 0.9 and 1.0 for mutation, and a length between 20 and 250 for 2-points crossover. Some results are shown in table 1 and represent data averaged over 10 independent runs. They are compared with the results obtained with the sequential execution of *WSAT*.

The main advantages of the PCGA method using a Moore's neighborhood over *WSAT* consists in a better convergence of the algorithm. The population keeps a diversity as the search proceeds. This is confirmed

Table 1. PCGA and WSAT on hard random 3-SAT problems.

variables	clauses	number of iterations (pop. size 320)	number of iterations (pop. size 2400)	number of iterations (WSAT)
64	275	16	12	68
128	550	35	23	131
256	1100	540	136	1356
512	2201	941	421	2242

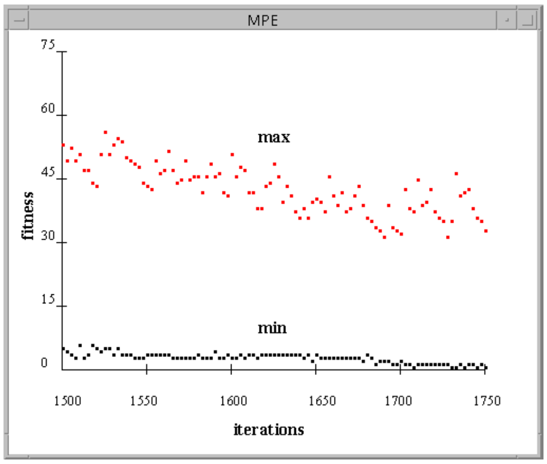

Figure 5. Fitness values.

by the difference between the maximum and minimum value of the fitness before the PCGA reaches the solution as is shown in figure 5. Such a difference is significantly narrowed in proximity of the solution. Before finding the solution the algorithm crosses a stationary zone, as shown in figure 5. A suitable tuning of the parameters of the algorithm should allow to reduce such stationary zone and obtain a faster convergence.

These initial experiments are very encouraging. They show that increasing the size of the population results in a better convergence, although this requires a major computational cost. This means that the parallel implementation of the method allows a good scalability on big size population to solve satisfiability problems having a large number of variables.

5. Conclusions

The paper presented a series of initial results regarding a new strategy for using a parallel cellular genetic algorithm to solve SAT problems. A parallel cellular automata environment has been used as a framework to implement the PCGA algorithm on a CS-2 parallel machine. The initial experiments have shown that PCGA has a better convergence than WSAT method. Our future work will be focused on improving the performance and the convergence of the method to solve more large problems.

References

[1] Cannataro M., Di Gregorio S., Rongo R., Spataro W., Spezzano G., Talia D., A Parallel Cellular Automata Environment on Multicomputers for Computational Science, *Parallel Computing*, North Holland, vol. 21, n. 5, pp. 803-824, 1995.

[2] Chang, C., Lee, R.C. *Symbolic Logic and mechanical theorem proving*. Academic Press, 1973.

[3] Cook, S.A., Mitchell, D. Finding hard instances of the satisfiability problem: a survey, *DIMACS series in Discrete Mathematics*, 1991.

[4] Garey, M.R., Johnson, D.S. *Computers and Intractability. A guide to the theory of NP-completeness*. San Francisco, Freeman, 1979.

[5] Gu, J. Global Optimization for Satisfiability (SAT) Problem. *IEEE Transaction on Knowledge and Data Engineering* 6(3) 361-381,1994.

[6] Hao J. K., Dorne R. A New Population-Based Method for Satisfiability Problems. *Proc. of ECAI 94*, 1994.

[7] De Jong A., Spears M.W. Using Genetic Algorithms to Solve NP-Complete Problems. *Proc. of the Intern. Conf. On Genetic Algorithms*, George Mason University, Fairfax, Virginia, June 1989.

[8] Kamath, A.P.,Karmarkar, N.K., Ramakrishnan, K.G., Resende, M.G.C., A continuos approach to inductive inference. *Mathematical Programming 57* 215-238, 1991.

[9] Kautz, H.A., Selman, B. Planning as satisfiability. *Proc. of ECAI-92*, 1992.

[10] Larrabee, T. Test pattern generation using Boolean satisfiability. *IEEE Trans. on Computer-Aided Design*,4-15, 1992.

[11] Martin, W.N., Lienig, J., Cohoon, J.P. *Island (migration) models: evolutionary algorithms based on punctuated equilibria*. Handbook of Evolutionary Computation, IOP Publishing, 1997.

[12] Mitchell, D., Selman, B., Levesque, H. Hard and easy distributions of SAT problems: *Proc. of AAAI*, 1992.

[13] Pettey, C. *Diffusion (cellular) models*. Handbook of Evolutionary Computation, IOP Publishing, 1997.

[14] Pizzuti C., Spezzano G., Ursino D. A Cellular Genetic Algorithm for Satisfiability Problems. In *Advances in Intelligent Systems*, Morabito F.C. (ed.), IOS Press, Amsterdam, 408-413, 1997.

[15] Selman, B., Levesque, H., Mitchell, D. A New Methods for Solving Hard Satisfiability Problems, *Proc. of AAAI*, 1992.

[16] Selman, B., Kautz, H.A., Cohen, B. Noise strategies for Improving Local Search, *Proc. of AAAI*, 1994.

[17] Spezzano G., Talia D., A High-level Cellular Programming Model for Massively Parallel Processing, *Proc. of the 2nd Int. Workshop on High-Level Programming Models and Supportive Environments HIPS'97*, IEEE Computer Society Press, pp. 55-63, April 1997.

[18] Toffoli T., Margolus N. *Cellular Automata Machines A New Environment for Modeling*. The MIT Press, Cambridge, Massachusetts, 1986.