Combining Cellular Genetic Algorithms and Local Search for Solving Satisfiability Problems

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

A new parallel hybrid method for solving the satisfiability problem that combines cellular genetic algorithms and the random walk (WSAT) strategy of GSAT is presented. The method, called CGWSAT, uses a cellular genetic algorithm to perform a global search on a random initial population of candidate solutions and a local selective generation of new strings. Global search is specialized in local search by adopting the WSAT strategy. CGWSAT has been implemented on a Meiko CS-2 parallel machine using a two-dimensional cellular automaton as parallel computation model. The algorithm has been tested on randomly generated problems and some classes of problems from the DIMACS test set.

1. Introduction

The satisfiability problem (*SAT*) consists in finding a truth assignment that makes a boolean expression true. Satisfiability plays a central role in a broad range of fields such as artificial intelligence, mathematical logic, computer vision, VLSI design, databases, automated reasoning, computer-aided design and manufacturing [9, 10, 12]. Many practical NP-complete problems [5] can be represented by a boolean formula in conjunctive normal form (*CNF*) and formulated as *SAT* problems, thus the development of efficient (exact and heuristics) methods for *SAT* can be exploited for solving combinatorial optimization problems. Among the proposed methods those based on local search [7, 17, 18] have received a lot of attention because they have been successfully applied to certain hard classes of large satisfiability problems [2]. 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. Satisfiability 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. 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. One of the most popular local search method for solving *SAT* is *GSAT* [17]. 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. An extension of GSAT, referred to as *random walk SAT* (*WSAT*), has been realized with the purpose of escaping from local optima by making upwards moves that could increase the number of unsatisfied clauses. In the last few years many different approaches based on simulated annealing, neural networks, genetic algorithms, tabu search, hybrid techniques that mix more than one heuristics [8, 3, 19, 20] have been proposed and compared with GSAT. Moreover, to significantly increase the size of SAT problems that can be solved, a variety of parallel processing techniques have been developed and efficient implementations have been realized [15].

In this paper we present a new parallel hybrid method for solving *SAT* problems that combines *cellular genetic algorithms* (*GA*) and the *WSAT* strategy. The method, called *Cellular Genetic WSAT* (*CGWSAT*), uses a cellular *GA* to carry out a global search on a random initial population of candidate solutions. New strings are generated by crossing the current string with one of the neighbors, with respect to a defined neighborhood relation. The search is then specialized locally by adopting the *WSAT* strategy as mutation operator. The integration of a cellular genetic algorithm and the *WSAT* approach revealed to be very successful, as experimental results show, because it takes advantages of the features of both methods. In fact, the cellular *GA* adopts a diffusion model of information among chromosomes by allowing the formation of subpopulations of strings having common characteristics inside the niches and relatively non competitive among them. Subpopulations diffuse information slowly thus avoiding to get trapped into local minima too early. However, the maintenance of a steady diversity in the population over the time is a very difficult task for a genetic algorithm. Diversity can be quickly lost during the recombination process. Increasing the population size does not help keep diversity. Lost of diversity undermines the effectiveness of recombination and, consequently, *GA* do not exhibit monotonic improvement. *CGWSAT*, by contrast, maintains a healthy diversity by adopting *WSAT* as mutation operator and by allowing to substitute the current string with one of the offspring, even if its fitness is worst than the father. A parallel version of the *CGWSAT* algorithm has been implemented on a *Meiko CS-2 parallel machine*. The algorithm has been tested on hard random *3-SAT* problems [14] and some classes of problems coming from the *2nd DIMACS Implementation Challenge* [11]. The experiments have been realized with the aim of comparing *WSAT* and *CGWSAT* performances with respect to the number of iterations needed to obtain a solution. In particular, as regard *WSAT* we present both the number of flips when the sequential algorithm runs and the number of flips when the algorithm is executed in parallel on the cellular automaton. The parallel execution of *WSAT* is, in fact, obtained when the probability of crossover in *CGWSAT* is set to zero. Thus we will show two kinds of results. The speedup of parallel *WSAT* with respect to sequential *WSAT*, and the speedup of *CGWSAT* with respect to both implementations of WSAT. The experimental results point out the very good outcomes of *CGWSAT* for all the tested problems.

The paper is organized as follows. Section 2 contains a detailed description of the WSAT algorithm. Section 3 contains a brief description of standard GA and a presentation of the cellular automata model to enable a fine-grained parallel implementation of GA through the diffusion model. Section 4 describes the new algorithm proposed. Section 5, finally, describes the parallel implementation of the method and discusses the experimental results.

procedure GSAT
Input: a set of clauses CL, and integers Maxflips and Maxtries.
Output: a satisfying truth assignment of CL, if any is found.
begin
  for i := 1 to Maxtries do
    T := a randomly generated truth assignment;
    for j := 1 to Maxflips do
      if T satisfies CL then return T;
      for each variable p:
        let Make[p] = the number of clauses
        currently unsatisfied by T that would
        become satisfied if the truth value of
        p were reversed (*flipped*).
        let Break[p] = the number of clauses
        currently satisfied by T that would
        become unsatisfied if the truth value
        of p were flipped.
        let Diff[p] = Make[p] - Break[p];
      end for
      let MaxDiffList = list of variables
        with the greatest Diff;
      p := a random member of MaxDiffList;
      T := T with the truth assignment
        of p flipped;
    end for
  end for
return "no satisfying assignment found";
end.

Figure 1. Basic GSAT algorithm.

2. The WSAT Procedure

WSAT [18] is an extension of GSAT [17] that mixes a random walk strategy with the greedy local search. It differs from GSAT in the selection of the variable to flip. It restricts the choice of a randomly flipped variable to the set of variables that appear in unsatisfied clauses. The basic GSAT 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 (Maxflips) is reached. This process is repeated as needed up to a maximum of Maxtries times. Figure 1 shows the basic GSAT algorithm.

GSAT is thus a greedy algorithm, that tries to flip variables so that as many clauses as possible are satisfied. Note that if the chosen variable p is such that $Diff[p] > 0$, then the total number of unsatisfied clauses decreases. This is called a downward move. If $Diff[p] = 0$, then the total number of satisfied clauses remains constant; this is called a sideways move. Finally, if the flipped variable has $Diff[p] < 0$, then an upward move is performed, in which the number of satisfied clauses decreases. Each iteration of the inner loop is referred to as a flip, and each iteration of the outer loop as a try.

The *random walk strategy* for satisfiability consists in flipping variables that appear in unsatisfied clauses. Several options are possible. In particular, 1) if no downward move is possible then with probability f flip any variable that appears in an unsatisfied clause instead of picking from the MaxDiffList; 2) regardless of the number of unsatisfied clauses, make a random walk move instead of a greedy move with probability f. Kautz and Selman have experimented that the best performances are obtained by adopting the second option with probability f between 0.5-0.6. Random walk strategy is one of the fasted (though incomplete) implemented procedures for SAT, it can solve hard problems with thousands of variables in few seconds.

3. Cellular Genetic Algorithms

Genetic algorithms (GA) [6] are stochastic search algorithms that have successfully been applied to a variety of optimization problems. Unlike most other optimization procedures, GA maintain a constant-size population of individuals (set of solutions) that are competitively selected to generate new candidates for the global optima. A standard GA works on elements (*chromosomes*), generally encoded as bit strings, representing a random chosen sample of candidate solutions to a given problem. 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. The algorithm explores all the search space altering the selected elements by means of genetic operators (*crossover, mutation*) that forms new elements to be evaluated. 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 good candidates for effective parallelization since their computational model is based on a population of individuals evolving in parallel. Parallel implementations of GA involve two main approaches. The *island* model [13] and the *diffusion* model [16]. 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 in a grid and interacts only within a particular neighbourhood. Information slowly diffuses across the grid thus clusters of solutions are formed around different optima.

Cellular automata (CA) [22, 23] 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 the 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 behavior 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* can be designed associating with each cell of a CA two substates: one contains a chromosome and the other its fitness. At the beginning, for each cell, a chromosome is randomly generated and its fitness is evaluated. Then, at each generation, the transition function associated with a cell selects the chromosome with the best fitness in the neighborhood. The genetic operator of crossover is applied to the current string and the selected string and the offspring is evaluated. If one of them has a better fitness than the current string, it becomes the current string. After that, mutation is applied to the current string.

4. The CGWSAT method

The satisfiability problem can be directly mapped onto a population of chromosomes of length n by interpreting every bit string as an assignment of truth values to the set of n variables. The *i-th* bit represents the truth value of the *i-th* boolean variable: *true* if the bit value is 1, *false* otherwise. 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. The *CGWSAT* method maps the population of strings into a two-dimensional square lattice. Every cell (i, j) of the cellular automaton contains a chromosome s and can only interact with the set of cells that neighbor itself. This defines a neighborhood relation. Every string s in the population is thus mated with the element w, among the k neighbors, where k depends on the neighborhood relation chosen, with the best fitness. Each cell contains a transition function that defines the $CGWSAT$ algorithm.

The pseudocode of the $CGWSAT$ algorithm performed by every cell is shown in figure 2. The cellular genetic algorithm adopts a 2-dimensional toroidal grid, using the Moore neighborhood. The transition function of each cell of the CA is described and uses the $CARPET$ programming language [21]. CARPET is an extension of the C language which contains additional constructs to realize cellular algorithms. CARPET programs can be developed and executed in parallel by the CAMEL support environment [1]. An extensive description of such an environment and its implementation can be found in [4]. The definition part $cadef$ of a CARPET program 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 during the computation. $cadef$ part in figure 2 defines a two-dimensional CA, with radius equal to 1 and a state with two substates ($chromosome$, $fitness$) that can be individually accessed. 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 the variables defined in the neighbor statement. $Moore$ is an alias to refer a neighbor cell variable and *step* 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.

At the beginning a random population of chromosomes is generated and their fitness is evaluated (*step* 0). At each generation, the transition function associated with a cell selects an element (the *partner*) among the k neighbors, where k depends on the neighborhood relation chosen, having the best fitness. Then the genetic operators of *crossover* and *mutation* are applied.

Crossover is a two-points crossover and 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 of length d. d is a parameter of the method. The current string s is then substituted by either u or v, the one having the best fit-

```
cadef
{
  dimension 2 ;
  radius 1 ;
  state ( float chromosome, int fitness);
  neighbor Moore[8] ([0,-1]North,[-1,-1]NWest,
                     [-1,0]West, [-1,1]SWest),
                     [0,1]South, [1,1]SEast),
                     [1,0]East, [1,-1]NEast);
  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,NWest_fitness,
                        Weast_fitness, SWeast_fitness),
                        South_fitness, SEast_fitness),
                        East_fitness, NEast_fitness);
   cross = rand_float();
   if (cross > pcross)
    {
    crossover(chrom,Moore[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 2. Pseudo-code of the cellular genetic algorithm.

ness. This kind of crossover coupled with the selection of the mate based on the neighborhood has a twofold advantage: first of all it allows the formation of subpopulations of strings having common characteristics inside the niches and relatively noncompetitive among them. Every niche thus evolves with a sufficient grade of independence by exploring different portions of the search space. Furthermore subpopulations diffuse information among them very slowly. This strategy, on the one hand, avoids to bring the population towards a too high homogeneousness, which would mean to get trapped into local minima, on the other hand the interaction among the chromosomes helps each other to improve their fitness.

Bit mutation is done according to the mixed random walk strategy of [18]. With probability f the variable with the best decrease in the number of unsatisfied clauses is flipped and, with probability $(1 - f)$ a variable that appears in some unsatisfied clause is picked and its assignment value changed. Both mutation and crossover operators have the role of bringing the population towards the best individuals and, at the same time, to introduce new strings to maintain a good diversity in the individuals which constitute the population. In fact, they introduce uphill moves that could increase the number of unsatisfied clauses but which, at the same time, employ a mechanism to escape from local minima by allowing to generate strings belonging to completely different portions of the search space. Finally, 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 cellular genetic algorithm to reach a solution in a low number of generations, as experimental results, presented in the next section, point out. It is worth to note that when the probability of doing crossover is set to zero, the automaton can be seen as a parallel implementation of WSAT: every string contained in a cell independently evolves to find a solution. We will show that the presence of the crossover dramatically reduces the number of flips necessary to reach a solution.

5 Parallel implementation and results

The $CGWSAT$ method is naturally suitable for running on distributed-memory $MIMD$ machines. It can be implemented on these machines using the $SPMD$ (Single-Program Multiple-Data) model. According to the SPMD approach, 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). $CGWSAT$ has been implemented on a *Meiko CS-2(Computing Surface 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. The parallel computation environment is provided by the CAMEL system, a cellular programming environment that allows to run CARPET programs and provides a development environment for editing, compiling, configuring, executing, monitoring and visualizing the output of programs.

CGWSAT has been tested on hard randomly generated 3-SAT problems, in particular, tests with 256 up to 2048 variables have been considered, and some problems of the DIMACS test suite [11]. In our experiments, we used a population size of 320, a radius equal to 1, a probability between 0.1 and 0.5 for crossover and between 0.9 and 1.0 for mutation, and a length approximately equals to 10% the number of variables for 2-points crossover.

Results are shown in table 1 and represent data averaged over 10 independent runs.

The experiments have been realized with the aim of comparing $WSAT$ and $CGWSAT$ performances with respect to the number of iterations needed to obtain a solution. In particular, as regard $WSAT$ we present both the number of flips when the sequential algorithm runs and the number of flips when the algorithm is executed in parallel on the cellular automaton. The parallel execution of $WSAT$, as already said, is obtained when the probability of crossover in $CGWSAT$ is set to zero. Thus we show two kinds of results. The speedup of parallel $WSAT$ with respect to sequential $WSAT$, and the speedup of $CGWSAT$ with respect to both implementations of $WSAT$. The sequential version of $WSAT$ is that developed by Kautz and Selman and available on the network (version 35). It has been executed on a SUN workstation with two *200 MHz Sparc* processors by using the option *-walk 0.5* in order to choose the random walk strategy. The experimental results point out the very good outcomes of $CGWSAT$ with respect to both sequential and parallel $WSAT$ for all the tested problems. As the table shows, there is a dramatic decrease in the number of flips of parallel $WSAT$ with respect to sequential $WSAT$.

Table 1. Hard random 3-SAT problems and DIMACS test suite.

Problem	variables	Clauses	iterations Seq. WSAT	iterations Par. WSAT	iterations CGWSAT
3-SAT	256	1100	1642	230	160
3-SAT	512	2201	3486	717	396
3-SAT	1024	4403	21372	5088	1875
3-SAT	2048	8806	27027	8470	3100
ii32b3	348	5734	2149	358	237
ii32c3	279	3272	3707	432	195
ii32d3	824	19478	28202	1425	1635
ii32e3	330	5020	1878	342	192
Jnh201	100	800	628	60	40
Jnh204	100	800	10436	221	105
jnh207	100	800	12734	170	163
jnh210	100	800	1499	89	57
g250.15	3750	233965	35037	12523	11102
f1000	1000	4250	510962	61472	26902

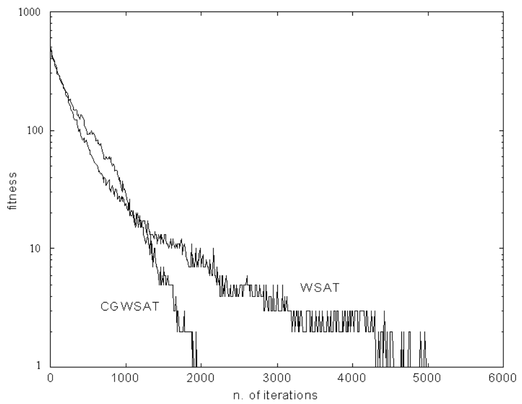

Figure 3. Fitness values of CGWSAT and parallel WSAT.

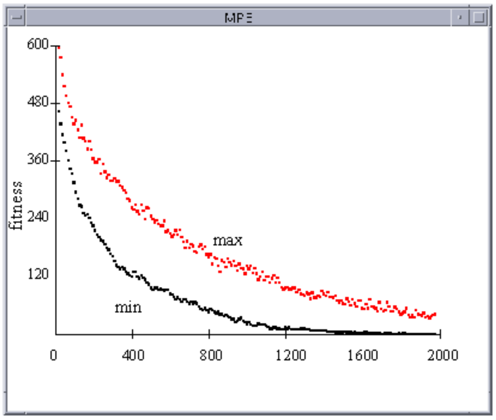

Figure 4. Fitness values.

The introduction of the crossover operator ulteriorly emphasizes such a decrease. This means that the diffusion of information inside the neighborhood is significant and brings *CGWSAT* to a better convergence. Such a behaviour is much more evident in figure 3 where the values of the fitness of both *CGWSAT* and parallel *WSAT* are shown during the computation for a 3-SAT problem with 1024 variables. The good performances of the method suggest the addition of the migration operator to allow the communication of different and distant portions of the search space. Finally, notice that the population keeps a good diversity as the search proceeds. This is confirmed by the difference between the maximum and minimum value of the fitness before the CGWSAT reaches the solution as is shown in figure 4.

6. Conclusions

The paper presented a series of results regarding a new strategy for using a parallel cellular genetic algorithms to solve *SAT* problems. A parallel cellular automata environment is used as a framework to implement the *CGWSAT* algorithm on a *CS-2* parallel machine. The experiments have shown that *CGWSAT* has a better convergence than *WSAT* method. Our future work will be focused on the introduction of the migration operator and on improving the performance and the convergence of the method to solve more large problems by an extensive study about the tuning of the parameters.

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

[3] 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.

[4] Folino, G., Pizzuti C., Spezzano G. Solving the Satisfiability Problem by a Parallel Cellular Genetic Algorithm. *Proc. of the 24th Euromicro Conference*, Vasteras, Sweden, August 1998.

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

[6] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing, 1989.

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

[8] Hao J. K., Dorne R. A New Population-Based Method for Satisfiability Problems. *Proc. of European Conf. on Artificial Intelligence (ECAI 94)*, 1994.

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

[10] Kautz, H.A., Selman, B. Planning as satisfiability. *Proc. of European Conf. on Artificial Intelligence (ECAI 92)*, 1992.

[11] Johnson, D.S., Trick, M.A. (eds.), *Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26, American Mathematical Society,1993.

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

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

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

[15] Pardalos, P. M., Pitsoulis, L., Resende, M.G.C. A Parallel GRASP for MAX-SAT Problems, *Workshop on Applied Parallel Computing in Industrial Problem Optimization*, Lyngby, Denmark, August 18-21, 1996.

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

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

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

[19] Spears, W. M. Using Neural Networks and Genetic Algorithms as Heuristics for NP-complete Problems, *Master Thesis, Dep. Computer Science, George Mason University, Virginia*, 1990.

[20] Spears, W. M. Simulated Annealing for Hard Satisfiability Problems, In Cliques, Coloring, and Satisfiability, *Second DIMACS Implementation Challenge*, David S. Johnson and Michael A. Trick (eds.), pp. 533-558, 1993.

[21] 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.

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

[23] Whitley, D. Cellular Genetic Algorithms, *Proc. Fifth Int. Conference on Genetic Algorithms*, 1993.