Word Equation Systems: The Heuristic Approach

César L. Alonso[1] *, Fátima Drubi[2], J. Gómez-García[3], and José Luis Montaña[3]

[1] Centro de Inteligencia Artificial, Universidad de Oviedo
Campus de Viesques, 33271 Gijón, Spain
calonso@aic.uniovi.es
[2] Departamento de Informática, Universidad de Oviedo
Campus de Viesques, 33271 Gijón, Spain
[3] Departamento de Matemáticas, Estadística y Computación,
Universidad de Cantabria
montana@matesco.unican.es

Abstract. One of the most intrincate algorithms related to words is Makanin's algorithm for solving word equations. Even if Makanin's algorithm is very complicated, the solvability problem for word equations remains NP-hard if one looks for short solutions, i. e. with length bounded by a linear function w. r. t. the size of the system ([2]) or even with constant bounded length ([1]). Word equations can be used to define various properties of strings, e. g. characterization of imprimitiveness, hardware specification and verification and string unification in PROLOG-3 or unification in theories with associative non-commutative operators. This paper is devoted to propose the heuristic approach to deal with the problem of solving word equation systems provided that some upper bound for the length of the solutions is given. Up to this moment several heuristic strategies have been proposed for other NP-complete problems, like 3-SAT, with a remarkable success. Following this direction we compare here two genetic local search algorithms for solving word equation systems. The first one consists of an adapted version of the well known WSAT heuristics for 3-SAT instances (see [9]). The second one is an improved version of our genetic local search algorithm in ([1]). We present some empirical results which indicate that our approach to this problem becomes a promising strategy. Our experimental results also certify that our local optimization technique seems to outperform the WSAT class of local search procedures for the word equation system problem.

Keywords: Evolutionary computation, genetic algorithms, local search strategies, word equations.

1 Introduction

Checking if two strings are identical is a rather trivial problem. It corresponds to test equality of strings. Finding patterns in strings is slightly more complicated.

* Partially supported by the spanish MCyT and FEDER grant TIC2003-04153

It corresponds to solve word equations with a constant side. For example:

$$xx01x1y = 010010101000110101010 \ . \tag{1}$$

where $x,\ y$ are variable strings in $\{0,1\}^*$. Equations of this type are not difficult to solve. Indeed many cases of this problem have very efficient algorithms in the field of pattern matching.

In general, try to find solutions to equations where both sides contain variable strings, like for instance:

$$x01x1y = 1y0xy \ . \tag{2}$$

where $x,\ y$ are variables in $\{0,1\}^*$ or show it has none, is a surprisingly difficult problem.

The satisfiability problem for word equations has a simple formulation: Find out whether or not an input word equation (like that in example (2)) has a solution. The decidability of the problem was proved by Makanin [6]). His decision procedure is one of the most complicated algorithms in theoretical computer science. The time complexity of this algorithm is $2^{2^{P(n)}}$ nondeterministic time, where $P(n)$ is a single exponential function of the size of the equation n ([5]). In recent years several better complexity upper bounds have been obtained: EXPSPACE ([4]), NEXPTIME ([8]) and PSPACE ([7]). A lower bound for the problem is NP ([2]). The best algorithms for NP-hard problems run in single exponential deterministic time. Each algorithm in PSPACE can be implemented in single exponential deterministic time, so exponential time is optimal in the context of deterministic algorithms solving word equations unless faster algorithms are developed for NP-hard problems.

In the present paper we compare the performance of two new evolutionary algorithms which incorporate some kind of local optimization for the problem of solving systems of word equations provided that an upper bound for the length of the solutions is given. The first strategy proposed here is inspired in the well known local search algorithms GSAT an WSAT to find a satisfying assignment for a set of clauses (see [9]). The second one is an improved version, including random walking in hypercubes of the kind $\{0,1\}^m$, of the flipping genetic local search algorithm announced in ([1]). As far as we know there are no references in the literature for solving this problem in the framework of heuristic strategies involving local search. The paper is organized as follows: in section 2 we explicitly state the WES problem with bounds; section 3 describes the evolutionary algorithms with the local search procedures; in section 4, we present the experimental results, solving some word equation systems randomly generated forcing solvability; finally, section 5 contains some conclusive remarks.

2 The Word Equation Systems Problem

Let A be an alphabet of constants and let Ω be an alphabet of variables. We assume that these alphabets are disjoint. As usual we denote by A^* the set of words on A, and given a word $w \in A^*$, $|w|$ stands for the length of w; ε denotes the empty word.

Definition 1. *A word equation over the alphabet A and variables set Ω is a pair $(L, R) \in (A \cup \Omega)^* \times (A \cup \Omega)^*$, usually denoted by $L = R$. A word equation system (WES) over the alphabet A and variables set Ω is a finite set of word equations $S = \{L_1 = R_1, \ldots, L_n = R_n\}$, where, for $i \in \{1, \ldots, n\}$, each pair $(L_i, R_i) \in (A \cup \Omega)^* \times (A \cup \Omega)^*$.*

Definition 2. *Given a WES over the alphabet A and variables set Ω, $S = \{L_1 = R_1, \ldots, L_n = R_n\}$, a solution of S is a morphism $\sigma : (A \cup \Omega)^* \to A^*$ such that $\sigma(a) = a$, for $a \in A$, and $\sigma(L_i) = \sigma(R_i)$, for $i \in \{1, \ldots, n\}$.*

The WES problem, in its general form, is stated as follows: given a word equation system as input find a solution if there exists anyone or determine the no existence of solutions otherwise. The problem we are going to study in this contribution is not as general as stated above, but it is also a NP-complete problem (see Theorem 5 below). In our formulation of the problem also an upper bound d for the length of the variable values in a solution is given. We name this variation the d-WES problem.

d-WES problem: Given a WES over the alphabet A with variables set Ω, $S = \{L_1 = R_1, \ldots, L_n = R_n\}$, find a solution $\sigma : (A \cup \Omega)^* \to A^*$ such that $|\sigma(x)| \leq d$, for each $x \in \Omega$, or determine the no existence otherwise.

Example 3. (see [1]) For each $d \geq 1$, let F_d and $WordFib_d$ be the d-th Fibonacci number and the d-th Fibonacci word over the alphabet $A = \{0, 1\}$, respectively. For any $d \geq 2$ let S_d be the word equation system over the alphabet $A = \{0, 1\}$ and variables set $\Omega = \{x_1, \ldots, x_{d+1}\}$ defined as:

$$x_1 = 0$$
$$x_2 = 1$$
$$01x_1x_2 = x_1x_2x_3$$
$$\ldots$$
$$01x_1x_2x_2x_3 \ldots x_{d-1}x_d = x_1x_2x_3 \ldots x_{d+1}.$$

Then, for any $d \geq 2$, the morphism $\sigma_d : (A \cup \Omega)^* \to A^*$, defined by

$$\sigma_d(x_i) = FibWord_i,$$

for $i \in \{1, \ldots, d+1\}$, is the only solution of the system S_d. This solution satisfies $|\sigma(x_i)| = F_i \leq F_{d+1}$, for each $i \in \{1, \ldots, d+1\}$. Recall that $FibWord_1 = 0$, $FibWord_2 = 1$ and $FibWord_i = FibWord_{i-2}FibWord_{i-1}$ if $i > 2$.

Remark 4. Example 3 is quite meaningful itself. It shows that any exact deterministic algorithm which solves the WES problem in its general form (or any heuristic algorithm solving all instances S_d) must have, at least, exponential worst-case complexity. This is due to the fact that the system S_d has polynomial size in d and the only solution of S_d, namely σ_d, has exponential length w.r.t d, because it contains, as a part, the d-th Fibonacci word, $WordFib_d$. Note that $WordFib_d$ has size equal to the d-th Fibonacci number, F_d, which is exponential w.r.t d.

A problem which does not allow to exhibit the exponential length argument for lower complexity bounds is the d-WES problem stated above. But this problem remains NP-complete.

Theorem 5. *(c. f. [1]) For any $d \geq 2$ the d-WES problem is NP-complete.*

3 The Evolutionary Algorithm

Given an alphabet A and some string over A, $\alpha \in A^*$, for any pair of positions i, j, $1 \leq i \leq j \leq |\alpha|$, in the string α, $\alpha[i,j] \in A^*$ denotes the substring of α given by the extraction of $j - i + 1$ consecutive many letters i through j from string α. In the case $i = j$, we denote by $\alpha[i]$ the single letter substring $\alpha[i,i]$, which represents the i-th symbol of the string α.

3.1 Individual Representation

Given an instance for the d-WES problem, that is, a word equation system $S = \{L_1 = R_1, \ldots, L_n = R_n\}$ with n equations and m variables, over the alphabet $A = \{0, 1\}$ and variables set $\Omega = \{x_1, \ldots, x_m\}$, if a morphism σ is candidate solution for S, then for each $i \in \{1, \ldots, m\}$, the size of the value of any variable x_i, $|\sigma(x_i)|$, must be less than or equal to d. This motivates the representation of a chromosome as a list of m strings $\{\alpha_1, \ldots, \alpha_m\}$ where, for each $i \in \{1, \ldots, m\}$, α_i is a word over the alphabet $A = \{0, 1\}$ of length $|\alpha_i| \leq d$, such that the value of the variable x_i, is represented in the chromosome by the string $\alpha_i \in A^*$.

3.2 Fitness Function

First, we introduce a notion of distance between strings which extends Hamming distance to the case of non-equal size strings. This is necessary because the chromosomes (representing candidate solutions for our problem instances) are variable size strings.

Given to strings $\alpha, \beta \in A^*$ the generalized Hamming distance between them is defined as follows:

$$H(\alpha, \beta) = Max\{|\alpha|, |\beta|\} - \sharp\{k \in \{1, \ldots, min\{|\alpha|, |\beta|\}\} : \alpha[k] = \beta[k]\}.$$

Given a word equation system $S = \{L_1 = R_1, \ldots, L_n = R_n\}$ over the alphabet $A = \{0, 1\}$ with set variables $\Omega = \{x_1, \ldots, x_m\}$ and a chromosome $\bar{\alpha} = \{\alpha_1, \ldots, \alpha_m\}$, representing a candidate solution for S, the fitness of $\bar{\alpha}$ is computed as follows:

First, in each equation, we substitute, for $j \in \{1, \ldots, m\}$, every variable x_j for the corresponding string $\alpha_j \in A$, and, after this replacement, we get the expressions $\{L_1(\bar{\alpha}) = R_1(\bar{\alpha}), \ldots, L_n(\bar{\alpha}) = R_n(\bar{\alpha})\}$ where $\{L_i(\bar{\alpha}), R_i(\bar{\alpha})\} \subset A^*$ for all $i \in \{1, \ldots, n\}$.

Then, the fitness of the chromosome $\bar{\alpha}$, $f(\bar{\alpha})$, is defined as:

$$f(\bar{\alpha}) = \sum_{i=1}^{n} H(L_i(\bar{\alpha}), R_i(\bar{\alpha})).$$

Proposition 6. *Let $S = \{L_1 = R_1, \ldots, L_n = R_n\}$ be a word equation system over the alphabet $A = \{0, 1\}$ with set variables $\Omega = \{x_1, \ldots, x_m\}$ and let $\bar{\alpha} = \{\alpha_1, \ldots, \alpha_m\}$ be a chromosome representing a candidate solution for S. Define the morphism $\sigma : (A \cup \Omega)^* \to A^*$ as $\sigma(x_i) = \alpha_i'$, for each $i \in \{1, \ldots, m\}$. Then the morphism σ is a solution of system S if and only if the fitness of the chromosome $\bar{\alpha}$ is equal to zero, that is $f(\bar{\alpha}) = 0$.*

Remark 7. According to Proposition 7, the goal of our evolutive algorithm is to minimize the fitness function f. By means of this fitness function, we propose a measure of the quality of an individual which distinguishes between individuals that satisfy the same number of equations. This last objective cannot be reached by other fitness functions like, for instance, the number of satisfied equations in the given system.

3.3 Genetic Operators

selection: We make use of the roulette wheel selection procedure (see [3]).

crossover: Given two chromosomes $\bar{\alpha} = \{\alpha_1, \ldots, \alpha_m\}$ and $\bar{\beta} = \{\beta_1, \ldots, \beta_m\}$, the result of a crossover is a chromosome constructed applying a local crossover to every of the corresponding strings α_i, β_i. Fixed $i \in \{1, \ldots m\}$, the crossover of the strings α_i, β_i, denoted as cr_i, is given as follows. Assume $a_i = |\alpha_i| \leq |\beta_i|$ then, the substring $cr_i[1, a_i]$ is the result of applying uniform crossover ([3]) to the strings $\alpha_i \in A^*$ and $\beta_i[1, a_i]$. Next, we randomly select a position $k_i \in \{a_i + 1, \ldots, d\}$ and define $cr_i[a_i + 1, k_i] = \beta_i[a_i + 1, min\{k_i, |\beta_i|\}\}]$.
We clarify this local crossover by means of the following example:

Example 8. Let $\alpha_i = 01$ and $\beta_i = 100011$ be the variable strings. In this case, we apply uniform crossover to the first two symbols. Let us suppose that 11 is the resulting substring. This substring is the first part of the resulting child. Then, if the selected position were, for instance, position 4, the second part of the child would be 00, and the complete child would be 1100.

mutation: We apply mutation with a given probability p. The concrete value of p in our algorithms is given in Section 4 below. Given a chromosome $\bar{\alpha} = \{\alpha_1, \ldots, \alpha_m\}$, the mutation operator applied to $\bar{\alpha}$ consists in replacing each gene of each word α_i with probability $\frac{1}{d}$, where d is the given upper bound.

3.4 Local Search Procedures

Given a word equation system $S = \{L_1 = R_1, \ldots, L_n = R_n\}$ over the alphabet $A = \{0, 1\}$ with set variables $\Omega = \{x_1, \ldots, x_m\}$ and a chromosome

$\bar{\alpha} = (\alpha_1, \ldots, \alpha_m)$, representing a candidate solution for S, for any $k \geq 0$ we define the k-neighborhood of $\bar{\alpha}$ with respect to the generalized Hamming distance as follows:

$$U_k(\bar{\alpha}) := \{\bar{\beta} : \ Maximum_{1 \leq i \leq m} \, H(\alpha_i, \beta_i) \leq k\}$$

Local search 1 (LS1) First, we present our adapted version of the local search procedure WSAT which will be sketched below. The local search procedure takes as input a chromosome $\bar{\alpha} = (\alpha_1, \ldots, \alpha_m)$ and, at each step, yields a chromosome $\bar{\beta} = (\beta_1, \ldots, \beta_m)$, which satisfies the following properties. With probability p, $\bar{\beta}$ is a random chromosome in $U_1(\bar{\alpha})$ and with probability $1 - p$, $\bar{\beta}$ is a chromosome in $U_1(\bar{\alpha})$ with minimal fitness. In this last case $\bar{\beta}$ cannot be improved by adding or flipping any single bit from $\bar{\alpha}$ (because their components are at Hamming distance at most one). This process iterates until a given specified maximum number of flips is reached. We call the parameter p probability of noise.

Below, we display the pseudo-code of this local search procedure taking as input a chromosome with m string variables of size bounded by d (one for each variable).

```
Input system S, chromosome cr, Maxflips M, probability p;
Procedure Local_Search1
begin
    for j=1 to M do
    begin
        if cr satisfies S then  return cr;
        cr1:= select (S,cr,p);
    end
    return cr1
end


Input system S, chromosome cr, probability p;
Procedure Select
begin
    u:= minimal fitness in U1(cr);
    if u=0 then
            cr1:= chromosome with minimal fitness in U1(cr)
    else
            with probability p:
                cr1:= a random chromosome in U1(cr);
            with probability 1-p:
                cr1:= chromosome with minimal fitness in U1(cr);
            end
    return cr1
end
```

Local search 2 (LS2) Suppose we are given a chromosome $\bar{\alpha} = (\alpha_1, \ldots, \alpha_m)$. At each iteration step, the local search generates a random walk inside the truncated hypercube $U_k(\bar{\alpha})$ and at each new generated chromosome makes a flip (or modifies its length by one unit if possible) if there is a gain in the fitness. This process iterates until there is no gain. Here k is the number of genes of the chromosome $\bar{\alpha}$, that is $k = \sum_{1 \leq i \leq m} |\alpha_i|$.

For each chromosome $\bar{\alpha}$ and each pair (i, j) such that, $1 \leq i \leq m$ and $1 \leq j \leq |\alpha_i|$ (representing the gene at position j in the i-component α_i of chromosome $\bar{\alpha}$) we define the set $U_{(i,j)}(\bar{\alpha})$ trough the next two properties:

- $U_{(i,j)}(\bar{\alpha}) \subset U_1(\bar{\alpha})$ and
- Any element $\bar{\beta} \in U_{(i,j)}(\bar{\alpha})$ satisfies: for all pair (i', j'), $1 \leq i' \leq m$; $1 \leq j' \leq |\alpha_i|$, if $(i', j') \neq (i, j)$ then $\alpha_{i'}[j'] = \beta_{i'}[j']$.

Note that any element in $U_{(i,j)}(\bar{\alpha})$ can be obtained in one of the following ways: if $j < |\alpha_i|$ by flipping the gene (i, j) in $\bar{\alpha}$; if $j = |\alpha_i|$ adding a new gene at the end of the component α_i of $\bar{\alpha}$, or deleting the gene (i, j) of the component α_i or flipping gene (i, j). In the pseudo-code displayed below we associate a gen g with a pair (i, j) and a chromosome cr with an element $\bar{\alpha}$. Then, notation $Ug(cr)$ denotes a subset of the type $U_{(i,j)}(\bar{\alpha})$.

```
Input chromosome cr;
Procedure Local_search2
begin
    H:=gens(cr);
    repeat
        cr_aux:=cr
        repeat
            g:= an element of H uniformly generated;
            cr:= chromosome with minimal fitness in Ug(cr);
            H:=H-{g}
        until empty H
    until cr=cr_aux
end
```

Summarizing, the pseudo-code of our evolutionary algorithms is the following:

```
begin
    Generation := 0;
    Population := initial_population;
    evaluate(Population);
    while (not_termination_condition) do
    begin
        Best := best_individual(Population);
        New_population := {Best};
        while (|New_population| < |Population|) do
```

```
        begin
            Pair := select_parents(Population);
            Child := crossover(Pair);
            Child := mutation(Child, probability);
            Child := local_search(Child);
            New_population := insert(Child, New_population);
        end
        Population := New_population;
        Generation := Generation + 1
    end
end
```

Remark 9. The initial population is randomly generated. The procedure evaluate(population) computes the fitness of all individuals in the population. The procedure local_search(Child) can be either LS1 or LS2. Finally, the termination condition is true when a solution is found (the fitness at some individual equals zero) or the number of generations attains a given value.

4 Experimental Results

We have performed our experiments over problem instances having n equations, m variables and a solution of maximum variable length q, denoted as pn-m-q. We run our program for various upper bounds of variable length $d \geq q$. Let us note that, m variables and d as upper bound for the length of a variable, determine a search space of size $(\sum_{i=0}^{d} 2^i)^m = (2^{d+1} - 1)^m$.

Since we have not found in the literature any benchmark instance for this problem, we have implemented a program for random generate word equation systems with solutions, and we have applied our algorithm to these systems [4].

All runs where performed over a processor AMD Athlom XP 1900+; 1,6 GHz and 512 Mb RAM. For a single run the execution time ranges from two seconds, for the simplest problems, to five minutes, for the most complex ones. The complexity of a problem is measured through the average number of evaluations to solution.

4.1 Probability of Mutation and Size of the Initial Population

After some previous experiments, we conclude that the best parameters for the LS2 program are **population size** equals 2 and **probability of mutation** equals 0.9. This previous experimentation was reported in ([1]). For the LS1 program we conclude that the best parameters are Maxflips equals 40 and probability of noise equals 0.2. We remark that these parameters correspond to the best results obtained in the problems reported in Table 1.

[4] Available on line in http://www.aic.uniovi.es/Tc/spanish/repository.htm

Table 1. Experimental results for various sizes of search space (S.S.). We have run for the instances the evolutionary algorithm with LS1 (SR1 & AES1) and with LS2 (SR2 & AES2). The elements of column U.B. are the different upper bounds.

P. instance	U.B.	S.S.	SR2	SR1	AES2	AES1
p10-8-3	3	2^{32}	100%	100%	3164.24	20930.6
p25-8-3	3	2^{32}	100%	100%	473.46	7304.34
p10-8-3	4	2^{40}	100%	100%	5121.25	41141.9
p25-8-3	5	2^{48}	100%	100%	1002.26	16824.5
p25-8-3	6	2^{56}	100%	100%	2193	25628.2
p5-15-3	3	2^{60}	100%	100%	16459.25	54459
p10-15-3	3	2^{60}	100%	100%	8124.48	55379.3
p15-12-4	4	2^{60}	100%	100%	479.63	4187.68
p10-8-3	7	2^{64}	100%	100%	168344	300204
p25-8-3	8	2^{72}	100%	100%	3567.86	69748
p10-8-3	10	2^{88}	85%	86%	405326	682543
p5-15-3	5	2^{90}	100%	96%	156365.52	546898
p10-15-3	5	2^{90}	100%	86%	258556	399374
p10-15-5	5	2^{90}	100%	100%	95457.93	180146
p25-23-4	4	2^{115}	100%	58%	412375	592463
p25-23-4	5	2^{138}	81%	34%	389630	858080
p15-25-5	5	2^{150}	98%	96%	278782.36	444430
p5-15-3	10	2^{165}	8%	0%	557410.58	–
p25-8-3	20	2^{168}	100%	78%	24221	689104

4.2 LS1 vs. LS2

We show the local search efficiency executing some experiments with both local search procedures. In all the executions, the algorithm stops if a solution is found or the limit of 1500000 evaluations is reached. The results of our experiments are displayed in Table 1 based on 50 independent runs for each instance. As usually, the performance of the algorithm is measured first of all by the *Success Rate (SR)*, which represents the portion of runs where a solution has been found. Moreover, as a measure of the time complexity, we use the *Average number of Evaluations to Solution (AES)* index, which counts the average number of fitness evaluations performed up to find a solution in successful runs. Comparing the two local search procedures, we observe that the improved version of our local search algorithm (LS2) is significantly better than the adapted version to our problem (LS1) of the WSAT strategies. This can be confirmed by looking at the respective Average number of Evaluations to Solution reported in our table of experiments. The comparatione between the evolutionary local-search strategy an the pure genetic approach was already reported in ([1]) using a preliminary version of (LS2) that does not use random walks. We observed there a very bad behavior of the pure genetic algorithm.

5 Conclusions, summary and future research

The results of the experiments reported in Table 1, indicate that the use of evolutive algorithms is a promising strategy for solving the d-WES problem, and that our algorithms have a good behavior also dealing with large search space sizes. Nevertheless, these promising results, there are some hard problems, as p5-15-3, over which our algorithms have some difficulties trying to find a solution and in other ones, as for example p25-8-3, the program always finds just the same. In both cases, the found solution agrees with that proposed by the random problem generator. In this sense, we have not a conclusion about the influence either of the number of equations or of the ratio size of the system/number of variables, on the difficulty of the problem. For the two compared local search algorithms we conclude that LS2 seems to outperform LS1, that is, the WSAT extension of local search procedures for the word equation system problem. Nevertheless it would be convenient to execute new experiments over problem instances with higher size of search space and to adjust for each instance, the parameters of Maxflips and probability of noise in procedure LS1. The most important limitation of our approach is the use of upper bounds on the size of the variables when looking for solutions. In a work in progress, we are developing an evolutionary algorithm for the general problem of solving systems of word equations (WES) that profits a logarithmic compression of the size of a minimal solution of a word equation via Lempel–Ziv encodings of words. We think that this will allow to explore much larger search spaces and avoiding the use of the upper bound on the size of the solutions.

References

1. Alonso C. L., Drubi F., Montana J. L.: An evolutionary algoritm for solving Word Equation Systems. Proc. CAEPIA-TTIA'2003. To appear in Springer L.N.A.I.
2. Angluin D.: Finding patterns common to a set of strings, J. C. S. S. 21(1) (1980) 46-62
3. Goldbert, D. E.: Genetic Algorithms in Search Optimization & Machine Learning. Addison Wesley Longmann, Inn. (1989)
4. Gutiérrez, C.: Satisfiability of word equations with constants is in exponential space. in Proc. FOCS'98, IEEE Computer Society Press, Palo Alto, California (1998)
5. Koscielski, A., Pacholski, L.: Complexity of Makanin's algorithm, J. ACM 43(4) (1996) 670-684
6. Makanin, G.S.: The Problem of Solvability of Equations in a Free Semigroup. Math. USSR Sbornik 32 (1977) 2 129-198
7. Plandowski, W.: Wojciech Plandowski: Satisfiability of Word Equations with Constants is in PSPACE. FOCS'99 495-500 (1999)
8. Plandowski, W., Rytter, W.: Application of Lempel-Ziv encodings to the Solution of Words Equations. Larsen, K.G. et al. (Eds.) L.N.C.S. 1443 (1998) 731-742
9. Selman, B., Levesque H., Mitchell: A new method for solving hard satisfiability problems. Pro. of the Tenth National Conference on Artificial Intelligence, AAAI Press, California (1992) 440-446