Observed Lower Bounds for Random 3-SAT Phase Transition Density Using Linear Programming

Marijn Heule* and Hans van Maaren

Department of Software Technology,
Faculty of Electrical Engineering,
Mathematics and Computer Sciences,
Delft University of Technology
marijn@heule.nl
h.vanmaaren@ewi.tudelft.nl

Abstract. We introduce two incomplete polynomial time algorithms to solve satisfiability problems which both use Linear Programming (LP) techniques. First, the FlipFlop LP attempts to simulate a Quadratic Program which would solve the CNF at hand. Second, the Weighted-LinearAutarky LP is an extended variant of the LinearAutarky LP as defined by Kullmann [6] and iteratively updates its weights to find autarkies in a given formula. Besides solving satisfiability problems, this LP could also be used to study the existence of autark assignments in formulas. Results within the experimental domain (up to 1000 variables) show a considerably sharper lower bound for the uniform random 3-Sat phase transition density than the proved lower bound of the myopic algorithm (> 3.26) by Achlioptas [1] and even than that of the greedy algorithm (> 3.52) proposed by Kaporis [5].

1 Introduction

Although Linear Programming (LP) techniques exist which run in polynomial time, it must be said that these techniques certainly are not among the most popular tools in the Satisfiability area. In those cases where they are used they are mainly applied in a preprocessing phase, in order to detect a specific structure of the CNF at hand. Examples of practical applications in this field using LP techniques are measuring the complexity of a CNF formula [2] and the detection of so-called equivalence clauses [9]. The reason why LP techniques are rarely used as a reasoning engine is that the straightforward LP relaxation of the Satisfiability Problem is always feasible (when unit clauses are absent). This means that in any trial to make LP techniques useful for CNF reasoning one has to come

* Supported by the Dutch Organization for Scientific Research (NWO) under grant 617.023.306.

F. Bacchus and T. Walsh (Eds.): SAT 2005, LNCS 3569, pp. 122–134, 2005.

up with a sophisticated reformulation. In this paper we introduce two LP based techniques which both result in a polynomial time running incomplete solver.

The first one, the FlipFlop LP, finds its motivation in the fact that the Satisfiability Problem can straightforwardly be reformulated as a Quadratic Programming Problem. We simulate this Quadratic Program through an iterated sequence of LP's, where the optimal solution of the n-th iteration is used as an input for the $n+1$-th iteration. That is, we reformulate the Quadratic Program as a "Contraction" problem, although not in the strict Mathematical sense. Satisfying assignments are among the "Contraction" fixed points, but unfortunately, the zero solution, which reveals no information at all, is also one of those fixed points. Hence, in order to make things work, we have to invoke certain tricks, which could be classified as "artificial", to stay away from this trivial fixed point.

The second LP is based on the detection of autarkies - partial assignments that satisfy all clauses that are "touched". We call it the WeightedLinear-Autarky LP, which finds its origin in Kullmann's Linear Autarky detection [6]. Also in this case we introduce in fact a sequence of LP's in order to find suitable weights for autarky detection. Again, we have to invoke tricks and some tuning to obtain a satisfactory performance.

Both of the above methods are generally applicable, but here we want to focus on their performance on uniform random 3-Sat formulas. From the presented results it becomes evident that the second approach is monotonically better than the first on these set of instances. Nevertheless we present both techniques, since we observed that outside the domain of random 3-Sat the situation is sometimes reversed.

The search for lower bounds for the uniform random 3-Sat phase transition density has a rich history by now [3]. To our knowledge the best result found by myopic algorithms is due to Achlioptas [1], but the greedy algorithm of Kaporis *et al* [5] pushes this bound a bit further. We emphasize that this contribution does not claim to come up with an even sharper proven lower bound. In order to do so one needs algorithms which are subject to some profound (likely probabilistic) analysis and LP techniques do not easily subject themselves as such. At least, such an analysis is beyond the capacity of the authors at this stage.

The question we address here has a more modest objective: to what observed densities can polynomial time algorithms of a given complexity solve uniform random 3-Sat problems. The experimentally obtained results show that both of our LP based techniques have the potential to go well above Kaporis' bound. We find this interesting as such, but most of all we take these results as an indication that LP based techniques are useful tools in the Satisfiability area.

As far as presenting the capacity of our methods within the context of uniform random 3-Sat and relating our results to the lower bound question, one might object that local search based solvers easily almost reach the phase transition density and that because of this feature our methods are situated in a sort of a "twilight zone": no analysis possible at the moment and outperformed by local search methods. To some extent this is indeed the case, however the authors are not aware of the existence of a local search solver with a default scalable

parameter setting resulting in an incomplete polynomial time solver reaching the phase transition density consistently. In case they exist, we hope to be alerted by the SAT community, and complexity comparisons must be made.

2 Preliminaries

A formula (denoted as $\mathcal{F} = c_1 \wedge c_2 \wedge \ldots \wedge c_m$) in *Conjunctive Normal Form* (CNF) is a conjunction of clauses; each clause (denoted as $c_i = l_1 \vee l_2 \vee \cdots \vee l_k$) being a disjunction of literals; and each literal is an atomic Boolean variable x_j or its negated form $\neg x_j$. The satisfiability (SAT) problem deals with the question whether a CNF formula is Satisfiable (has a Boolean solution) or not. A formula is satisfiable if an assignment satisfies all clauses. A clause is satisfied if at least one of its literals is satisfied.

The *clause-variable matrix* A associated to a CNF formula $\mathcal{F}$ is the matrix defined by

$$A_{i,j} = \begin{cases} 1 & \text{if the } i^{th} \text{ clause of } \mathcal{F} \text{ contains the } j^{th} \text{ variable with sign 1} \\ -1 & \text{if the } i^{th} \text{ clause of } \mathcal{F} \text{ contains the } j^{th} \text{ variable with sign -1} \\ 0 & \text{otherwise.} \end{cases}$$

Thus if $\mathcal{F}$ is a CNF formula with propositional variables $x_1, \ldots, x_n$ the SAT problem for $\mathcal{F}$ reads as the -1,1 Feasibility problem

$$\begin{cases} Ax \geq -L + 2e \\ x \in \{-1, 1\}^n \end{cases}$$

In the above, L is the *length vector*, having the length of clause i as its i^{th} entry, and e is the all one vector of appropriate dimension. Notice that we use {-1,1} Boolean variables instead of the commonly used {0,1}.

As defined by Kullmann [6], a formula with clause-variable matrix A has a Linear Autarky $x \in \mathcal{Q}^n$ if

$$\begin{cases} Ax \geq 0 \\ x \neq 0 \end{cases}$$

The above concept generalizes an earlier version of Warners and van Maaren [10] which provides a decomposition of a formula in case the kernel of its clause-variable matrix is non-zero.

A so-called *monotone variable* is the best known example of a Linear Autarky: if variable x_j appears only positive (negative) in the formula, vector $x = e_j$ ($x = -e_j$), (e_j is the j^{th} unit vector), is a Linear Autarky. In general, a Linear Autarky x leads to an autark partial assignment of the formula involved by rounding. To examine this, let partial assignment sign be

$$\mathsf{sign}(x_j) = \begin{cases} 1 & \text{if } x_j > 0 \\ -1 & \text{if } x_j < 0 \\ \text{undefined} & \text{if } x_j = 0. \end{cases}$$

and substitute all defined $\mathsf{sign}(x_j)$ into the formula. If a clause i is affected by this substitution, that is, if a variable with defined sign occurs in clause i, this clause is obviously satisfied by the partial assignment since

$$\sum_j A_{ij}x_j = \sum_{x_j \neq 0} A_{ij}x_j \geq 0$$

and hence not all $A_{ij}x_j$ with defined $\mathsf{sign}(x_j)$ can be negative. The above implies that a Linear Autarky leads to a non-trivial decomposition of the formula at hand. One part is satisfied by sign and the other part contains only variables undefined by sign. Clearly, the latter part is Satisfiability Equivalent to the original formula: it is satisfiable if and only if the original formula is.

3 The FlipFlop LP

A Quadratic Program (QP) formulation of the SAT problem as defined in the preliminaries is shown in Fig. 1(a). Since max $\sum x_i^2$ is a quadratic objective function (not a linear), solving this QP could not be executed in polynomial time, unless P = NP. We propose a linear simulation of this QP that is incomplete - in contrast to the QP - but could solve CNF formulas in polynomial time. This linear simulation is shown in Fig. 1(b). We use parameters v_i that represent "guessed" outcomes of x_i in max $\sum x_i^2$.

$$\begin{array}{ll} \max & \sum x_i^2 \\ \text{s.t.} & \begin{cases} Ax \geq -e \\ -1 \leq x_i \leq 1 \end{cases} \end{array} \qquad \begin{array}{ll} \max & \sum v_i x_i \\ \text{s.t.} & \begin{cases} Ax \geq -e \\ -1 \leq x_i \leq 1 \end{cases} \end{array}$$

(a) (b)

Fig. 1. (a) A 3-SAT QP formulation; (b) A possible linear simulation of (a)

Solving difficult problems requires a close approximation of x_i by v_i. We experimented with initial weights $v_i := \sum_{x_i \in \mathcal{F}} - \sum_{\neg x_i \in \mathcal{F}}$. This LP could solve uniform random 3-SAT instances up to density of approximately 2.5.

To stretch this result, we experimented with various update strategies for the weights v_i. One rather intuitive update strategy applies the outcomes of x_i in the solution of max $\sum v_i x_i$ as an "educated guess" for the weights in a new iteration: $v_i^{n+1} := \mathrm{R}^n(x_i)$ with $\mathrm{R}^n(x_i)$ referring to the value of x_i in the solution of the linear simulation after iteration n. Even if one allows many iterations, this LP is unable to solve many problems apart from the ones that could be solved using the initial weights.

Of the various update strategies we used during our experiments, one clearly solved the most problems. The recipe: divide the variables in two disjunct sets $\mathcal{A}$ and $\mathcal{B}$. Variables were randomly divided between sets $\mathcal{A}$ and $\mathcal{B}$ with an equal

$$x_i \in \begin{cases} \mathcal{A} & p(0.5) \\ \mathcal{B} & \text{otherwise} \end{cases} \qquad v_i^{n+1} := \begin{cases} \mathrm{R}^n(x_i) & \text{if } x_i \in \mathcal{A} \textbf{ and } n \equiv 0 \ (mod\ 2) \textbf{ or} \\ & \text{if } x_i \in \mathcal{B} \textbf{ and } n \equiv 1 \ (mod\ 2) \\ 0 & \text{otherwise} \end{cases}$$

(a) (b)

Fig. 2. (a) Initial set definition; (b) Update strategy

probability. The weights are updated in such a way that the weights become 0 if the corresponding variable is alternately in set $\mathcal{A}$ or set $\mathcal{B}$. This update strategy attempts to assign variables - as many as possible - in one set to the sign of the corresponding weights. This results in assigning values $\neq 0$ to variables in the other set. The $\mathrm{R}^n(x_i)$ values will be used in iteration $n+1$ where the same process is performed, but with swapped sets (see Fig. 2). We refer to iteratively solving the linear simulation defined in Fig. 1(b) using the update strategy as defined in Fig. 2 as the FlipFlop LP.

4 Weighted Linear Autarkies

In this section, we explain the concept of *weighted linear autarkies* and the corresponding LP. A proper weight matrix is required to solve CNF's. First we show how LP techniques can be used to determine the importance of the variables. Second, we propose an update strategy which use this information to construct the weight matrix.

4.1 The Weighted Linear Autarky LP

We refer to the *weighted clause-variable matrix* W associated to a CNF formula $\mathcal{F}$ as the weighted analogue of matrix A using weights $w_{i,j} > 0$:

$$W_{i,j} = \begin{cases} w_{i,j} & \text{if the } i^{th} \text{ clause of } \mathcal{F} \text{ contains the } j^{th} \text{ variable with sign 1} \\ -w_{i,j} & \text{if the } i^{th} \text{ clause of } \mathcal{F} \text{ contains the } j^{th} \text{ variable with sign -1} \\ 0 & \text{otherwise.} \end{cases}$$

The definition of the LinearAutarky LP as proposed by Kullmann [6] is shown in Fig. 3(a). Our variant uses the weighted clause-variable matrix W (see Fig. 3(b)). We refer to this LP as the WeightedLinearAutarky LP. For both LP's, one can try to achieve the constraint $x \neq 0$ by an objective function. Two possible objective functions are shown in Fig. 4.

$$\begin{cases} Ax \geq 0 \\ x \neq 0 \end{cases} \qquad \begin{cases} Wx \geq 0 \\ x \neq 0 \end{cases}$$

(a) (b)

Fig. 3. (a) LinearAutarky LP; (b) WeightedLinearAutarky LP

$$\max \quad \sum x_i \qquad\qquad \max \quad \sum v_i x_i \quad v_i := \sum_{x_i \in \mathcal{F}} - \sum_{\neg x_i \in \mathcal{F}}$$

(a) (b)

Fig. 4. (a) Elementary objective function; (b) Advanced objective function

The advanced objective function seemed more effective, since it finds all monotone variables directly. This in contrast to the elementary objective function which will not find monotone variables that occur only negative. However, during our experiments we found no differences in the solving capacity of both objective functions. Therefore we selected the elementary one.

As with the LinearAutarky LP, the set of variables with $x_i \neq 0$ in the solution of a WeightedLinearAutarky LP forms an autark assignment. The advantage of the WeightedLinearAutarky LP is that it could - in theory - solve many problems. The following theorem is quite similar to Lemma 3.2 by Kullmann in [7].

Theorem 1. *For every satisfiable formula $\mathcal{F}$, there exists a weighted clause-variable matrix W of which its* WeightedLinearAutarky *LP results in a solution.*

Proof: Given a satisfying assignment of $\mathcal{F}$, construct W in such a way that satisfied literals $x_{i,j}$ have corresponding weight $w_{i,j} = L_i - 1$ and falsified literals $x_{i,j}$ have corresponding weight $w_{i,j} = 1$. L_i refers to the length of clause i. The WeightedLinearAutarky LP will result in the given satisfying assignment. □

In practice, it is hard to construct a W leading to a solution. We propose a method to construct matrix W using Linear Programming techniques. First, we solve a LP called the UpdateWeights LP, which is quite similar to the WeightedLinearAutarky LP. Second, we use the values of variables x_i in the solution of that UpdateWeights LP to construct W by applying an *update strategy.*

4.2 The Update Weights LP

We saw that solving the LP with constraints $Wx \geq 0$ and $x \neq 0$ would result in an autark assignment if it is feasible. The LP with constraints $Wx+y \geq 0$, $x \neq 0$, and $y > 0$ is always feasible, but would rarely result in an autark assignment. However, solutions for this LP could be useful to update W in such a way that the LP with constraints $Wx \geq 0$ and $x \neq 0$ becomes feasible. This idea is the foundation of the UpdateWeights LP as defined in figure 5.

The values of parameters y_{MIN} and y_{MAX} are essential for the solving capacity of the algorithm. Since the variables x_i are unbounded, there is only one parameter that needs to be measured: $\frac{y_{\text{MAX}}}{-y_{\text{MIN}}}$. One must choose y_{MIN} and y_{MAX} in such a way

$$\max \quad \sum y_i \qquad\qquad \begin{cases} Wx + y \geq 0 \\ x \neq 0 \\ y_{\text{MIN}} \leq y_i \leq y_{\text{MAX}} \end{cases}$$

(a) (b)

Fig. 5. UpdateWeights LP definitions: (a) objective function; (b) constraints

that in as many relaxations of clauses at least one $x_{i,j}$ exists with $\mathsf{sign}(x_{i,j}) = 1$. If $\frac{y_{\text{MAX}}}{-y_{\text{MIN}}}$ is too large, many clauses will have either three literals with $\mathsf{sign}(x_{i,j}) = 1$ or three literals with $\mathsf{sign}(x_{i,j}) = -1$. On the other hand, if $\frac{y_{\text{MAX}}}{-y_{\text{MIN}}}$ is too small, many variables will have an undefined $\mathsf{sign}(x_{i,j})$. Throughout our experiments on uniform random 3-Sat formulas, $\frac{y_{\text{MAX}}}{-y_{\text{MIN}}} \approx 2$ appeared to solve most instances near the phase transition density. More specific, we used y_{MIN} := -0.3 and $y_{\text{MAX}} := 0.6$.

4.3 Update Strategy

We refer to $\mathrm{S}^n(x_i)$ as the value of x_i in the solution of the UpdateWeights LP after iteration n. We want to use these values $\mathrm{S}^n(x_i)$ to construct matrix W^{n+1}. An update strategy must satisfy the definition property that all $w_{i,j} > 0$. Notice that $\mathrm{S}^n(x_i)$ is unbounded, since variables x_i are unbounded in the UpdateWeights LP. Now it seems natural to make $w_{i,j}^{n+1} := base^{S^n(x_i)w_{i,j}^n}$. To prevent that weights might reach either 0 or ∞, we introduce two parameters w_{MIN} and w_{MAX} which refer to the minimum and the maximum of the weights, respectively. This results in the following strategy:

$$\begin{cases} W^0 = A \\ w_{i,j}^{n+1} = base^{S^n(x_i)w_{i,j}^n} \\ 0 < w_{\text{MIN}} \leq w_{i,j} \leq w_{\text{MAX}} \end{cases}$$

Fig. 6. Update strategy for the WeightedLinearAutarky LP

Parameter w_{MIN} has no significant influence on the performance of the algorithm as long as its value is small. We chose $w_{\text{MIN}} := 0.01$. A small performance gain could be achieved by making $w_{\text{MAX}} \approx 5$ (assuming y_{MIN} = -0.3 and $y_{\text{MAX}} = 0.6$). We did some small scale experiments to determine an effective value for parameter *base*: using the other parameters on their chosen values as mentioned above, we experimented with uniform random 3-Sat formulas. We generated 1000 instances on density 3.7 using 200 variables. The influence of parameter *base* on the performance was measured starting with $base = 1$ using steps of 0.25 and from 4.0 we used only integer values.

Results are shown in Fig. 7. The optimal value $base = 2.5$ shown here is used during our further experiments. Notice that for $base = 1$ the WeightedLinearAutarky LP equals the Linear Autarky LP. So using the LinearAutarky LP, we cannot solve one single sample of these instances.

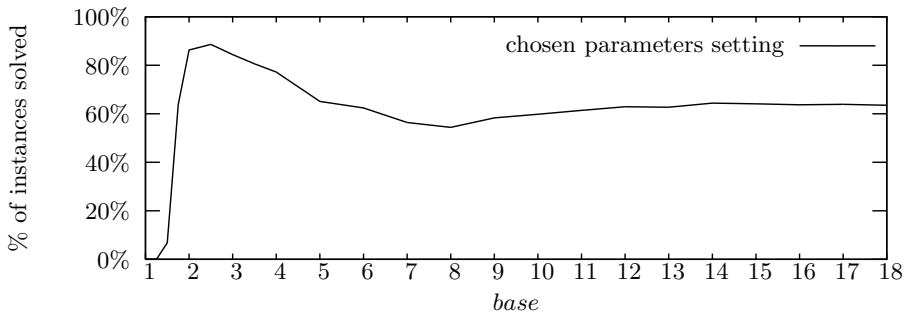

Fig. 7. Influence of parameter *base* on the solving capacity of the algorithm

Although intensively tweaking of the various parameters might result in a better performance of the algorithm, we used these settings for our experiments.

5 Experimental Results

This section deals with the performance of the two solvers based on the LP's described above. Our goal is to observe a lower bound for the uniform random 3-SAT phase transition density. During the experiments we used an Intel Pentium 4 with 3.5 GHz running on Fedora Core 2. All LP's were solved using glpsol from the GNU Linear Programming Kit[1] (GLPK). To generate the uniform random 3-SAT formulas, we used the generator of Van Gelder[2] [8].

As we will see, the WEIGHTEDLINEARAUTARKY LP clearly performs better than the FLIPFLOP LP on uniform random 3-SAT formulas. The reason for presenting the results of the latter is the fact that for some structured problems we observed a reverse effect. Presentation of these results is outside the scope of this paper.

Since Linear Programming is a polynomial time solving procedure, the complexity of the presented LP's is also polynomial. But only, of course, as long as the required iterations to solve a formula does not grow exponentially with respect to its size. We used a small constant to limit the number of iterations (MAX_ITERATIONS := 50). Using this constant, we already reached a point where, as the number of variables increased, the density on which all instances were solved increased accordingly. A higher constant can only increase the performance of the solvers using the presented LP's. This will be explained later. For reasons of comparison we added the results of Kaporis' algorithm on the same samples (see Fig. 12) at the end of this section.

[1] available at `http://www.gnu.org/software/glpk/glpk.html`
[2] available at `http://www.satlib.org`

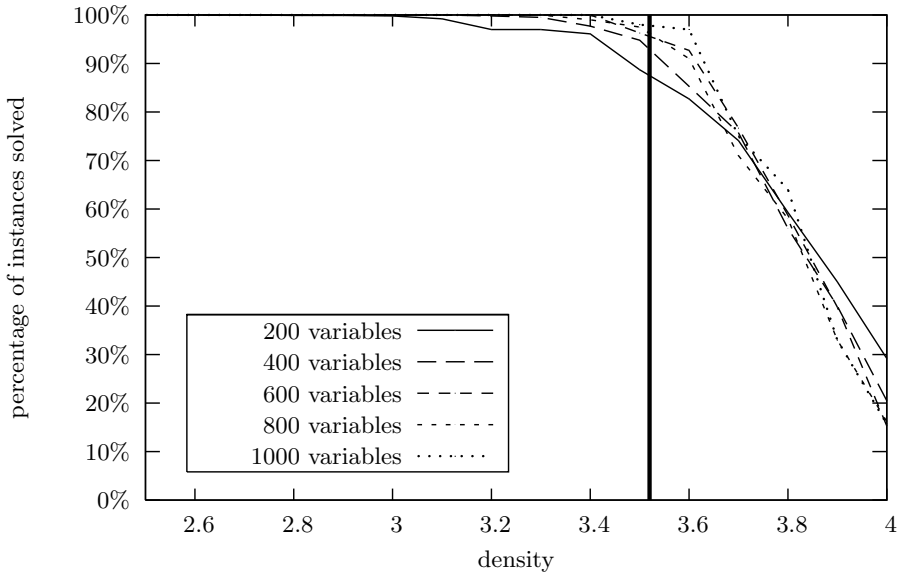

Fig. 8. Percentage of instances solved using the FlipFlop LP. Vertical line shows the proved lower bound by Kaporis [5]

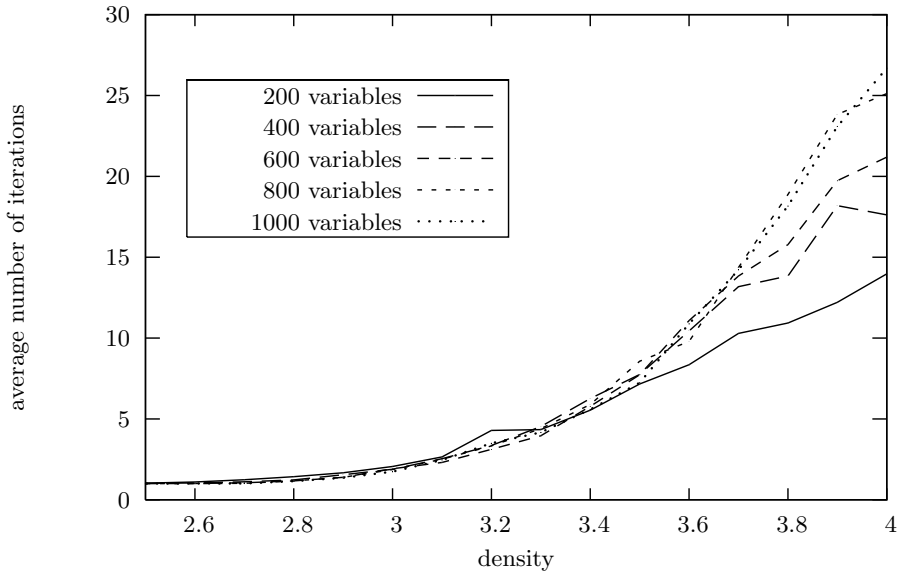

Fig. 9. Average number of iterations required by the FlipFlop LP

5.1 Flip Flop Solver

The FlipFlop solver (see algorithm 1) first attempts to assign all variables to {-1,1} by using initial weights based on the occurrences of the variables in

$\mathcal{F}$. In the first iteration these weights are used to solve the linear simulation in Fig. 1(b). For all next iterations, it uses the FlipFlop update strategy. A solution is found when all variables are assigned a value {-1,1} by the LP.

Algorithm 1 FlipFlop solver($\mathcal{F}$)

```
1: for i ∈ {1, .., n} do
2:    v_i := Σ_{x_i ∈ F} − Σ_{¬x_i ∈ F}
3: end for
4: for j ∈ {1, .., MAX_ITERATIONS} do
5:    v := FlipFlopLP(A_F, v, j modulo 2)
6:    if Σ_{i=1}^{n} v_i^2 = n then
7:       return SATISFIABLE
8:    end if
9: end for
10: return UNKNOWN
```

We experimented for each density (using steps of 0.1) with 1000 instances of 200 variables, 600 instances of 400 variables, 400 instances of 600 variables, 200 instances of 800 variables and 100 instances of 1000 variables. Figure 8 shows the percentage of instances solved by the FlipFlop solver. We observed that by increasing the number of variables, higher densities are reached where all instances were solved. However, it consequently lags behind the WeightedLinearAutarky solver.

Figure 9 shows the average number of iterations required to find a solution of the solved instances. Recall that we terminate the solver after 50 iterations. Up to density 3.0 these averages are slightly lower than the averages by the WeightedLinearAutarky solver (see for comparison Fig. 11). The FlipFlop solver owes its performance to the many instances that were solved in the first iteration - due to the initial weights.

5.2 Weighted Linear Autarky Solver

The WeightedLinearAutarkySolver, as shown in algorithm 2, attempts to solve a formula $\mathcal{F}$ by alternately solving the WeightedLinearAutarky LP and the UpdateWeights LP. We used the update strategy as defined above to construct W in each iteration.

The WeightedLinearAutarkyLP($W_{\mathcal{F}}$) returns set $\mathcal{I}$, which contains all variables with $x_i \neq 0$. If $\mathcal{I} \neq \emptyset$, then the LP found an autark assignment. All clauses that contain at least one variable in $\mathcal{I}$ are removed from the formula. This is denoted by IterativeUnitPropagation($\mathcal{F} \cap \mathcal{I}$). If no clause is left in $\mathcal{F}$ after this removal, the formula is satisfiable.

As with the FlipFlop solver, we experimented for each density with 1000 instances of 200 variables, 600 instances of 400 variables, 400 instances of 600 variables, 200 instances of 800 variables and 100 instances of 1000 variables (using steps of 0.1). Figure 10 shows the percentage of instances solved by the

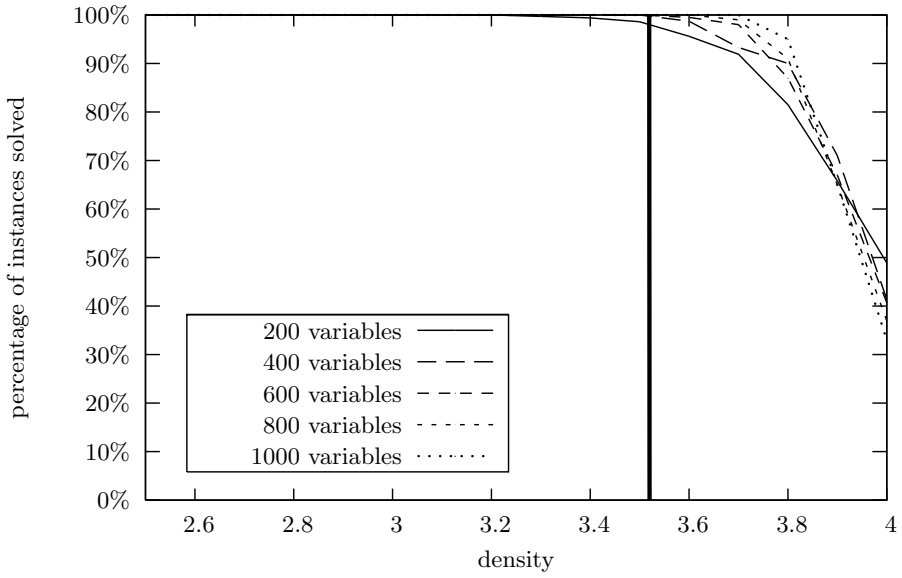

Fig. 10. Percentage of instances solved using the WeightedLinearAutarky LP. Vertical line shows the proved lower bound by Kaporis [5]

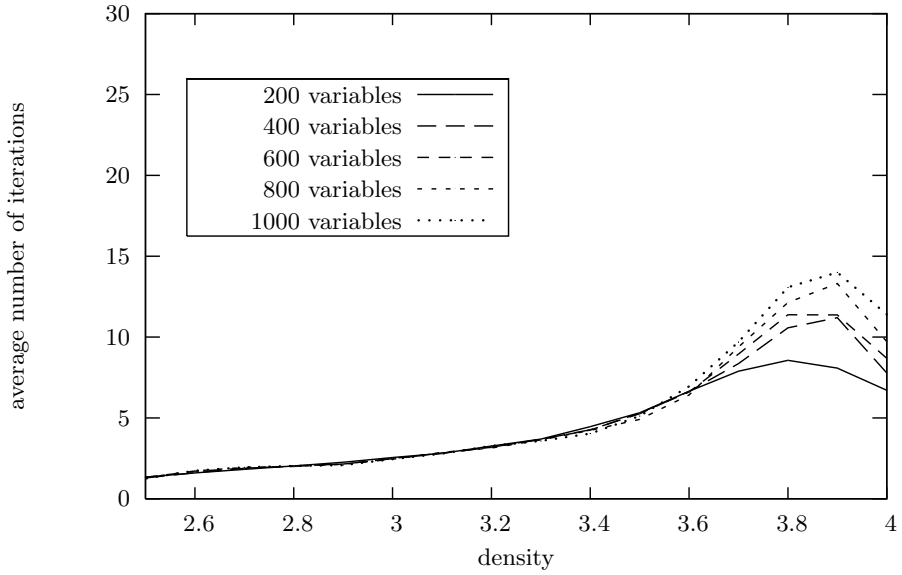

Fig. 11. Average number of iterations required by the WeightedLinearAutarky LP

Algorithm 2 WeightedLinearAutarkySolver($\mathcal{F}$)

```
1: W_F := A_F
2: for j ∈ {1, .., MAX_ITERATIONS} do
3:    I := WeightedLinearAutarkyLP(W_F)
4:    if I ≠ ∅ then
5:       F := IterativeUnitPropagation(F ∩ I)
6:       W_F := A_F
7:       if F = ∅ then
8:          return SATISFIABLE
9:       end if
10:   else
11:      W_F := UpdateWeightsLP(W_F)
12:   end if
13: end for
14: return UNKNOWN
```

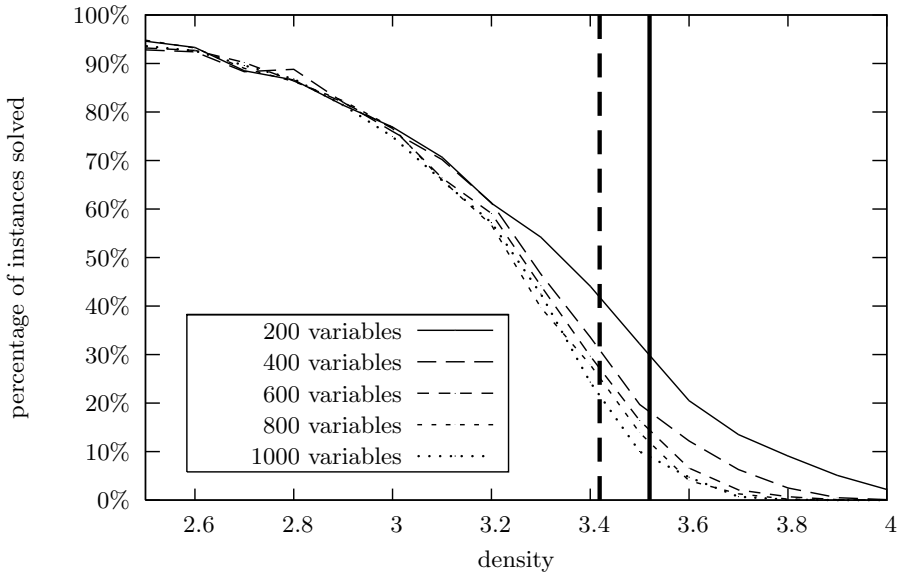

Fig. 12. Performance of the greedy algorithm proposed by Kaporis [4]. Dashed vertical line shows the proved lower bound by Kaporis in [4]; straight vertical line shows the proved lower bound by Kaporis in [5]

WeightedLinearAutarky solver. With an increasing number of variables come higher densities where all instances were solved: all instances consisting of 200 variables on density 3.4 were solved, while on density 3.7 all instances consisting of 1000 variables were solved. Figure 11 shows the average number of iterations required to find a solution of the solved instances. Although there is a small increase in this average - while increasing the number of variables - the number of iterations divided by the number of variables is decreasing.

During our experiments, a vast majority of the found autark assignments were either monotone variables or satisfying assignments. Up to density 3.4, we found some autark assignments consisting of multiple literals that only satisfied a part of the formula. However, this only occurred in approximately 0.5% of the formulas, regardless of the number of variables.

6 Conclusions

Within the experimental domain, both our methods show a threshold shaped success performance ratio of proving Satisfiability with respect to increasing density, growing steeper with increasing size. In this context, our second method shows a better performance. If the reader allows us to extrapolate to infinity the observed success ratio of 100% seems to be located well above the Kaporis' bound. For reasons of comparison we implemented the greedy algorithm of Kaporis described in [4] and investigated its performance on the same samples. The emerged figure 12 shows that this algorithm has to catch up quite a bit within the range $[1000, \infty]$. From its decreasing performance with growing size (within our domain of experimentation) above the Kaporis' bound one could guess that for very large size instances this success performance ratio must have a perfect threshold shape. Our main conclusion however is that LP techniques, having a modest polynomial complexity, apparently are able to reveal hidden structure in CNF's, even in case of randomly generated formulas. The interesting aspects lie in the fact that they are not evidently resolution based and that their reasoning mechanisms, which clearly must be there, are rather unconventional compared to the standard ones used in the Satisfiability area.

References

1. D. Achlioptas, *Lower Bounds for Random 3-SAT via Differential Equations.* Theoretical Computer Science **265**(1-2) (2001), 159–185.
2. E. Boros, Y. Crama, P. Hammer, and M.Saks, *A complexity Index for Satisfiability Problems.* SIAM Journal on Computing **23** 1 (1994), 45–49
3. J. Franco, *Results related to threshold phenomena research in Satisfiability: lower bounds.* Theoretical Computer Science **265**(1-2) (2001), 147–157.
4. A.C. Kaporis, L.M. Kirousis, E.G. Lalas, *The Probabilistic Analysis of a Greedy Satisfiability Algorithm.* Lecture Notes In Computer Science **2461** (2002), 574–585.
5. A.C. Kaporis, L.M. Kirousis, E.G. Lalas, *Selecting complementary pairs of literals.* Electronic Notes in Discrete Mathematics **16** (2003).
6. O. Kullmann, *Investigations on autark assignments.* Discrete Applied Mathematics **107**(1-3) (2000), 99–137.
7. O. Kullmann, *Lean clause-sets: generalizations of minimally unsatisfiable clause-sets.* Discrete Applied Mathematics **130**(2) (2003), 209–249.
8. A. Van Gelder, Problem generator `mkcnf.c` contributed to the DIMACS Challenge archieve.
9. J.P. Warners, H. van Maaren, *A two phase algorithm for solving a class of hard satisfiability problems.* Oper. Res. Lett. **23**(3-5) (1998), 81-88.
10. J. Warners and H. van Maaren, *Solving satisfiability problems using elliptic approximations. Effective branching rules.* Discrete Applied Mathematics **107**(1-3) (2000), 241–259.