# ***Finding a satisfying assignment for semi-random satisfiable 3CNF formulas***

**Thesis for the M.Sc. degree**

**Dan Vilenchik**

**Under the Supervision of Professor Uriel Feige**
Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science

Submitted to the Scientific Council of the
Weizmann Institute of Science
Rehovot, Israel 76100.

January 2004

## Contents

**Abstract**

The "planted sat" distribution is the set of satisfiable $3CNF$ formulas generated in the following manner. First pick a truth assignment, then include each clause satisfied by it w.p. $p$. For $p = \frac{d \log n}{n^2}$ ,a majority vote procedure reconstructs w.h.p the original assignment for large enough constant $d$. For $p = \frac{d}{n^2}$, $d$ a large enough constant, an algorithm presented by Flaxman (published in *SODA* 2003) solves the problem w.h.p, thus proving a conjecture by Papadimitriou and Koutsoupias from 1992 regarding sparse instances. We present a modification to that algorithm and evaluate both algorithms in a semi-random model. While Flaxman's algorithm fails in the semi-random model, our modification makes the algorithm resilient to the adversary we define.

# 1 Introduction

A $3CNF$ formula over the variables $x_1, x_2, ..., x_n$ is a conjunction of clauses $C_1, C_2, ..., C_m$ where each clause is a disjunction of three literals. Each literal is either a variable or its negation. A $3CNF$ is satisfiable if there is a boolean assignment to the variables s.t. every clause contains at least one satisfied literal. $3SAT$ is the language of all satisfiable $3CNF$ formulas.

Although $2SAT$ is known to be in $P$, $3SAT$ is one of the most famous $NP$-complete problems ([16]). Understanding the complexity of the Satisfiability problem on various natural instances is a fundamental problem of computer science. The Satisfiability problem appears in many basic applications such as automatic deductive reasoning, logic programming, machine learning, etc. Thus, while the problem is believed unsolvable in polynomial time in the worst case, many heuristical algorithms have been proposed and some of them seem to perform quite good "on average".

An interesting algorithmic question, is understanding the hardness of *finding* a satisfying assignment for a random $3CNF$. Notice that this is not exactly the same as *deciding* the satisfiability of an instance. We may have an algorithm that finds an assignment with good probability, but when such an algorithm fails, we cannot use that information to decide the satisfiability of the formula.

Algorithmic theory of random graphs has been the focus of extensive research in the recent years (see [23] for a comprehensive survey of the area). As part of this trend, uniformly random $3CNF$s (generated by selecting at random $\Delta n$ clauses over the variables $\{x_1, ..., x_n\}$) has also caught the eyes of many researchers. This distribution was investigated in many diverse fields such as physics [27], combinatorics [22], proof complexity [15], algorithm analysis [2] and hardness of approximation [18], to mention just a few. It is known that there is a sharp threshold in the ratio of clauses to variables ([22]); a random $3CNF$ with clause-to-variable ratio below the threshold is satisfiable w.h.p and one with ratio above the threshold is unsatisfiable w.h.p. This threshold is not know exactly (and not even known to be independent of $n$). The threshold is known to be at least 3.42 ([24]) and at most 4.5 ([25]). Experimental results predict the higher end of the interval ([17]).

For random $3CNF$s, it was proved by Broder, Frieze and Upfal [12], that the pure literal heuristic finds w.h.p a satisfying assignment for random formulas with clause to variable ratio of $\Delta = 1.63$ (note that for such a ratio the formula is almost surely satisfiable). Ben-Sasson and Alekhnovich show in [4] that the random walk procedure finds w.h.p a satisfying assignment for formulas of that ratio in polynomial time. Kaporis, Kirousis and Lalas give in [24] a greedy Davis-Putnam algorithm that finds w.h.p in polynomial time a satisfying assignment for random $3CNF$s with clause to variable ratio of 3.42 (their analysis is based on the method of differential equations. For more details on this method see [1]).

In this work we concentrate on formulas with constant (above the threshold) clause-to-variables ratio. At such ratios, most formulas are unsatisfiable, and analyzing the distribution of random satisfiable instances seems hard. Thus, we focus on the planted-sat distribution.
There are two variants of the planted sat distribution. In both cases we start by picking at random a truth assignment $\psi$. In one variant we pick at random $cn$ clauses satisfied by $\psi$, in the other we include in the instance each clause satisfied by $\psi$ w.p $d/n^2$ (where the natural relation is $c = 7d/6$). We use the second variant and give a formal definition later on.
The distribution is highly interesting in its own right. It has been studied empirically in [13], and is the analog of the planted clique, planted bisection, and planted coloring distributions studied e.g. in [6],[20],[5],[8]. The planted sat distribution is studied in [29], where an algorithm, based on greedy variable assignment rule, successfully discovers a satisfying assignment w.h.p for dense instances ($p = \Theta(1/n)$). The authors conjecture that some modification of their algorithm will work in sparser instances (namely, in a constat ratio). The algorithm presented in [21] proves this conjecture. The planted distribution with ratio $c \log n$ is studied in [7] to show that the "democratic" algorithm (the majority vote procedure as we call it) produces a satisfying assignment w.h.p (when $c$ is a large enough constant). In [4], it is proved that the random walk procedure takes w.h.p time exponential in $n$ in finding a satisfying assignment for planted instances with a clause to variable ratio greater than $\Delta_0$ ($\Delta_0$ is a large enough constant).

While the known results on planted $3CNF$s imply that most of them are easy to solve, planted formulas tend to be of a very special type. For example, w.h.p most variables of a random planted $3CNF$ appear in roughly the same number of clauses. Moreover, most variables have a gap between the number of true appearances and false ones. So, formulas created in a "somewhat random" manner may not be satisfied by algorithms for the random case. On the other hand, worst-case assumptions may be overly pessimistic in many situations. To explore the intermediate range, and a wider variety of satisfiable $3CNF$ instances, semi-random models are used.

Semi random models generate problem instances by blending random and adversarial decisions. In the semi-random model, such as studied in [8],[19], after a random instance is generated (according to the defined distribution), an adversary is allowed to further modify the instance. These modifications cannot be arbitrary, or else the adversary can turn the instance into an "uninteresting" worst case instance. Thus, we restrict the adversary in the way he can modify the instance.

We now survey some known results in planted semi-random models. Consider the problem of *coloring* with up to $k$ colors a $k$-colorable graph. A common model for generating planted $k$-colorable graphs is first partitioning the set of $n$ vertices into $k$ disjoint sets of size $n/k$. Then, for every two vertices in different sets, include an edge between them w.p. $p$. For random planted $k$-colorable graphs, Blum and Spencer give in [8] an algorithm which w.h.p $k$-colors $k$-colorable graphs whenever $p > n^{\epsilon-1}$ for any constant $\epsilon > 0$. Alon and Kahale give in [5] an algorithm (based on spectral technique) that 3-colors planted 3-colorable graphs w.h.p for $p \geq d/n$, $d$ a large enough constant.

Blum and Spencer, in [8], describe two semi-random variants of the random planted coloring model, based on the semi-random source of Santha and Vazirani ([31]). In the first variant, an adversary makes its choices for the graph edges, but each choice is flipped with some small probability ("noise"). In the second variant, a random graph is chosen first, and then an adversary can modify the graph subject to some restrictions. This last variant was formalized by Feige and Kilian [19] as the *sandwich model*. For the semi-random model, Blum and Spencer give in [8] a probabilistic algorithm that colors 3-colorable graphs w.h.p for edge probability $p \geq n^{-0.6+\epsilon}$. Feige and Kilian [19] give a coloring algorithm (based on finding independent sets using semi-definite programming) for $p = (1+\epsilon)k \ln n/n$, improving the result of Blum and Spencer. The algorithm of Alon and Kahale fails in the semi-random model.

We now move to the *clique* problem. Alon, Krivelevich and Sudakov [6] designed an algorithm based on spectral techniques that w.h.p finds a clique of size $\Omega(\sqrt{n})$ hidden in an otherwise random graph. Feige and Krauthgamer [20] show a different algorithm, based on the Lovász theta function, that both finds the hidden clique and certifies its optimality. While the first algorithm fails in the semi-random hidden clique model (in which an adversary can remove edges from the random part of the graph), the second algorithm is resilient to the adversary.

We return to the work of Feige and Kilian on semi-random models ([19]) in the context of *independent sets* and *graph bisections*. To generate a semi-random graph with a planted independent set, an independent set of $\alpha n$ vertices is randomly chosen. Each edge connecting $S$ and $\overline{S}$ is chosen w.p $p$. The adversary may add edges as long as $S$ remains an independent set. Feige and Kilian give in [18] a heuristic that w.h.p recovers an independent set of size $\alpha n$ whenever $p > (1+\epsilon)\log n/\alpha n$. In the semi-random model for graph bisection, a random bisection $(S, \overline{S})$ of the vertices is chosen. Each edge crossing the bisection is chosen w.p $q$, and each edge not crossing is chosen w.p $p \geq q$. The adversary can then remove edges in $(S, \overline{S})$ and add edges not in $(S, \overline{S})$. Extending the work of Boppana [11] (for the random model) to the semi-random model, Feige and Kilian give a heuristic that recovers this bisection w.h.p whenever $p - q \geq c\sqrt{p \log n/n}$ for large enough $c$.

In all the works above (except Blum and Spencer) the algorithms for the random models were based on eigenvalue approach, and the extension to the semi-random model was based on semi-definite programming.

Lastly, we consider the *Hamiltonian path* problem. Let $G$ be a random graph. It was proved [28] that if $G$ has at least $\frac{n}{2}(\log n + \log\log n)$ edges, and it has minimum degree 2, then $G$ is Hamiltonian

w.h.p. Several polynomial time algorithms were suggested to find w.h.p Hamiltonian cycles in such graphs. Among them is the algorithm HAM, suggested by Bollobás, Fenner and Frieze [10].
The semi-random variant of this problem is as follows. First, a random graph which has at least $\frac{n}{2}(\log n + \log\log n)$ edges, and has minimum degree 2 is generated. Then, an adversary may add as many edges as he wishes. While the algorithm given in [10] encounters problems in the semi-random setting, an algorithm given by Keydar [26] (which is an adaptation of the latter algorithm to the semi-random model) solves these problems.

In our work we defined the following semi-random variant of the planted sat distribution. First a random instance is generated in the planted distribution. Let $\varphi$ be the planted satisfying assignment. We say that a clause is *3-satisfiable* w.r.t some assignment $\psi$ if all its three literals evaluate to true under $\psi$. Returning to our semi-random model, after a random instance is generated, an adversary may add an arbitrary set of 3-satisfiable clauses w.r.t $\varphi$.
A few words as to why we chose this restriction are in place. 3-satisfiable clauses are "easy" for many natural heuristics for 3SAT (random walk, majority vote, etc ...). Moreover, If one could tell which clauses are adversary clauses, one could immediately deduce the assignment given to these clauses variables by $\varphi$. To sum up, adding 3-satisfiable clauses can be considered as planting "guiding tips" toward $\varphi$ in the instance. We can use this adversary as a benchmark to check whether an algorithm is, in some sense, natural, robust, not over- fitted to the model in which the instance is generated, etc.
Our main contribution is a probabilistic and a deterministic algorithms for finding a satisfying assignment for $3CNF$ formulas in the semi-random model described above. The general outline of the algorithm is taken from [21], however we try to give a more natural version of the algorithm, resulting in an algorithm that also works in the semi-random setting (unlike the one presented in [21]). The main heuristic we are using was first introduced by Alon and Kahale in the context of 3-coloring sparse 3-colorable graphs [5]. The heuristic was then adopted by Chen and Frieze for 2-coloring 2-colorable bipartite hypergraphs [14]. Lastly, Flaxman adopted the heuristic for the planted sat distribution (with a linear number of clauses) [21].

## 2 The Planted Sat Distribution

We now rigourously define the planted sat distribution. First we pick an assignment $\varphi$ ,uniformly at random, over the variables $\{x_1, x_2, ..., x_n\}$. Let $d$ be a large enough constant (as needed by the analysis). For each satisfiable clause (with respect to $\varphi$) include it with probability $p = \frac{d}{n^2}$. Let $I$ be the final instance we got.
Clearly $I$ is satisfiable, since at least $\varphi$ satisfies it.
The total number of clauses over the variables $\{x_1, x_2, ..., x_n\}$ is $\binom{n}{3} \cdot 2^3$ (note that we disregard the order of variables inside the clause). We have $\binom{n}{3}$ unsatisfiable clauses with respect to $\varphi$. All in all, we have $7 \cdot \binom{n}{3}$ clauses satisfied by $\varphi$. The expected size of $I$ is $p \cdot 7 \cdot \binom{n}{3} \approx \frac{7 \cdot d \cdot n}{6}$.

We will use the notation $\mathbb{P}_{d,n}$ to denote the planted distribution with parameter $p = d/n^2$ over $n$ variables. We will use the notation $I \in_R \mathbb{P}_{d,n}$ to denote a $3CNF$ instance, $I$, randomly picked from $\mathbb{P}_{d,n}$.

# 3 Algorithm in Detail

## 3.1 Some Definitions

A truth assignment to the variables $x_1, x_2, ..., x_n$ is a vector of length $n$ over $\{0, 1\}$. The i$'th$ entry is the assignment given to $x_i$. A partial truth assignment is a vector of length $n$ over $\{0, 1, *\}$, where $*$ stands for a variable which is not assigned.
We define the following properties for a variable $x$ with respect to an instance $I$ sampled from $\mathbb{P}_{d,n}$ and a partial truth assignment $\tau$.
We say that a variable $x$ *supports* a clause with respect to $\tau$ if it is the only variable to satisfy it, and the other two variables are assigned by $\tau$.
Every variable $x$ is expected to support $\binom{n-1}{2} \cdot \frac{d}{n^2} = \frac{d}{2} + O(\frac{1}{n})$ clauses.
Let $\epsilon$ be some small constant independent of $d$ (e.g. 0.01).
We say that $x$ is a *strong supporter* if $x$ supports at least $\frac{d}{2} - \epsilon d$ clauses.
We say that $x$ is a *mild supporter* if $x$ supports at least $\frac{d}{2} - 2\epsilon d$ clauses (in the expression $2\epsilon d$, 2 is an arbitrary constant greater than 1).
We say that a variable $x$ is *assigned correctly*, if its assignment agrees with $\varphi$, the original satisfying assignment of $I$. Otherwise, we say that the variable is *wrongly assigned* (even if it is assigned correctly with respect to some other satisfying assignment).

## 3.2 Some Intuition

First we give some intuition for the algorithm. Let $\varphi$ be the original satisfying assignment of $I$. W.l.o.g, let $x$ be a variable assigned true by $\varphi$. Look at the set of all clauses satisfied by $\varphi$ in which $x$ appears. We have more clauses in which it appears as a positive literal than as a negative one (since clauses, where all three literals are false, are excluded, while clauses where all three are true, are not). Each clause is included in $I$ independently with the same probability. Therefore, we expect to find in $I$ more clauses in which $x$ appears positively than clauses in which it appears negatively. Hence one can deduce quite a good approximation for $\varphi$, from a majority vote taken separately for every variable (over the clauses of $I$ in which it appears). In fact, when $\frac{\#clauses}{\#variables} > c \log n$ for $c$ large enough, one can reconstruct the original assignment w.h.p (see [7]). Naturally then, a majority vote would be the first step of our algorithm.

Then follows a step of unassignment. In this step we unassign variables which we suspect were wrongly assigned by the majority vote. We observe that every variable has in expectation a certain amount of clauses which it supports. We chose this as the criteria for unassignment (a less restricting and more natural criteria than [21]). Namely, if a variable has too few clauses which it supports, we unassign it.
We then simplify the formula (delete satisfied clauses, and erase false literals from unsatisfied ones) and complete the algorithm by a RandomWalkSat procedure on the subformula induced by unassigned variables.
RandomWalkSAT is a simple probabilistic procedure for finding a satisfying assignment (if such exists). It starts with a given assignment to the variables. In every step, we pick an unsatisfied clause and flip the assignment of a randomly picked variable in it. We go on until we reach a satisfying assignment or some bound on the number of flips.
For a random $3CNF$, w.h.p a RandomWalkSat procedure takes time exponential in the number of variables (see [4]). Although we have $O(n)$ unassigned variables, the formula induced by unassigned variables breaks into variable-disjoint subformulas, each containing at most $\log n$ variables. Due to this, the RandomWalkSat will almost surely find a satisfying assignment in polynomial time.

### 3.3 Flaxman's Algorithm

A full description and analysis of Flaxman's algorithm is given in [21]. For completeness, we give here a short description of the algorithm.
In his paper, Flaxman uses the same notation, $d$, for both the definition of the distribution space and the parametrization of the algorithm. We don't assume that the algorithm is given $d$, not in Flaxman's case nor in ours (the intuition will become clear when discussing the semi-random setting). Thus using $d$ for both defining $\mathbb{P}_{d,n}$ and describing the algorithm would be somewhat an abuse of notation. We use $d$ for the definition of $\mathbb{P}_{d,n}$ and $d'$ for the parametrization of the algorithm. When reading the algorithm, it would be convenient for the reader to assume that $d' = d$ (we will show later that w.l.o.g one can assume that $d' = d$).

The first phase is getting a good approximation for the original satisfying assignment. This is done using the following spectral technique (see also [5], [14]). The instance $I$ induces a graph $G(V, E)$ s.t. $V$ is the set of all $2n$ literals. For any two literals $l_i, l_j$, $(l_i, l_j) \in E$ if there exist a clause in $I$ containing both. Next, edges incident to vertices of degree greater than some constant (Flaxman uses $180d'$) are removed to receive $G'$. Let $A$ be the adjacency matrix of $G'$. Let $e$ be the eigenvector corresponding to the most negative eigenvalue of $A$. Denote by $e_i$ the i'th entry of $e$. The assignment $\pi$ that $e$ induces is $\pi(x_i) = T$ iff $e_i \geq 0$. The approximating assignment is $\pi$ or $\neg\pi$, depending which one satisfies more clauses of $I$.
A note on spectral approximation vs. the majority vote is in due. Flaxman looks at a more general distribution. In his model, every clause with exactly $i \in \{1, 2, 3\}$ literals satisfied by $\varphi$ is included

w.p. $p_i$. $p_1 = d/n^2$, and $p_i = \delta_i \cdot p$, $\delta_i \in [0,1]$ for $i = 2,3$. In that model, a majority vote does not necessarily give a good approximation (depending on $\delta_2, \delta_3$). For that reason spectral approximation techniques are used. If $p_1 = p_2 = p_3$ we can safely replace the spectral technique with a majority vote procedure (as also suggested in [21]).

Then comes a phase of improving the approximation we got. The improvement is done iteratively, in $\log n$ iterations. In every iteration, for every variable, if it appears in at least $5\epsilon d'$ unsatisfied clauses, its assignment is negated in the next iteration ($\epsilon$ is some constant of the algorithm similarly to our definitions).

Note that every variable is expected to appear in $3.5d + O(\frac{1}{n})$ clauses and to support $\frac{d}{2} + O(\frac{1}{n})$ clauses w.r.t to $\varphi$.
The next phase is an unassignment phase. First we unassign variables whose support (w.r.t the improved approximation) is not in the range $\frac{d'}{2} \pm 4\epsilon d'$, and variables which appear in more than $3.5d' + \epsilon d'$ clauses, or less than $3.5d' - \epsilon d'$. Then, while there is a variable appearing in less than $3.5d' - 2\epsilon d'$ clauses with all variables assigned, unassign it.

Completing the algorithm is an exhaustive search phase. Consider the graph $G$ whose vertices are all unassigned variables and we have an edge between two variables if they appear together in some clause. If any connected component of $G$ is larger than $\log n$ fail, otherwise complete the satisfying assignment for $I$ using an exhaustive search in every component.

*Remark:* As we mentioned above, Flaxman looks at a slightly different model (in particular, the order of literals inside a clause is not disregarded). When describing his algorithm we adjusted the expectations (whose values define the parameters of the unassignment phase) to fit our model (where the order of literals inside a clause is disregarded).

### 3.4 Our Algorithm

Let $c_0$ be the constant s.t. when the clause probability is $p \geq c_0 \log n/n^2$, w.h.p the majority vote is correct for all variables (see for example [7]). We repeat the following phases with $d' = 1 \ldots c_0 \log n$, or until we find a satisfying assignment.
*Remark:* for the time being, assume that $n$ is known to the algorithm. We show later how to overcome this technical problem.

The first phase is a majority vote. We assign each variable $x$, independently of the others, according to the majority vote on his sign (i.e. if it appears more as the literal $x$ than as $\overline{x}$ we assign it $T$, otherwise we assign it $F$). We break ties arbitrarily (e.g., assigning the variable $T$). Let $\psi$ be the assignment that was output after this phase. If $\psi$ satisfies $I$ return $\psi$.

In the second phase, we set $\psi_0 = \psi$. We define the following test for a variable in the $i$'$th$ iteration ($i \geq 1$): we check whether a variable supports at least $\frac{d'}{2} - 2\epsilon d'$ clauses w.r.t $\psi_{i-1}$.
We perform the test independently for every variable at the beginning of the iteration. If a variable

fails the test we unassign it in $\psi_i$.
We keep going until $\psi_i = \psi_{i+1}$ (i.e., until there are no more variables to unassign).
After the unassignment ends, we simplify the formula w.r.t the partial assignment we got.
By *simplify a formula* w.r.t a partial assignment, we mean remove all satisfied clauses, and in unsatisfied ones, remove literals which evaluate to false. Note that when a simplification procedure ends, we end up with clauses of length 1,2 and 3.

Next, we perform a *unit-clause-propagation* simplification. In a *unit-clause-propagation*, we go over all clauses of length 1 and assign them in a satisfying manner. We then simplify the formula. We repeat this procedure until there are no more clauses of length 1. If a contradiction is found (a clause of the form $x$ and a clause of the form $\overline{x}$) we fail the algorithm.

In the last phase, we perform a RandomWalkSAT procedure on the remaining clauses. Note that all remaining clauses contain only unassigned variables.
We first pick at random an assignment for the set of unassigned variables. Then, while there exists an unsatisfied clause, we pick one (while possible pick unsatisfied clauses of length 2 rather than 3; this increases the probability of a good move in the flip) and flip a randomly picked variable in it. We bound the total number of flips by $3tn$, where $t > n$ is a parameter of the algorithm controlling the failure probability of the random walk (which is at most $\frac{n}{\log n}e^{-t/n}$). If a satisfying assignment was not reached within $3tn$ steps, we fail the algorithm.

**Theorem 3.1.** *There exists a constant $\Delta_0$ s.t. for every $d \geq \Delta_0$, the algorithm outputs w.h.p a satisfying assignment for a semi random instance $I$ (as defined in section 7.1), whose random part is randomly picked from $\mathbb{P}_{d,n}$.*

# 4 Paper Outline

To prove theorem 3.1, we will first present several properties that a random formula in $\mathbb{P}_{d,n}$ has w.h.p. We will call a formula which has all those properties *typical*. Then, we will show that on a typical formula, the algorithm always succeeds in finding a satisfying assignment. We then repeat the same outline in the semi-random model.
The properties of a random instance are presented in section 5. In section 6 we prove that the algorithm works on a typical instance. The semi-random model is discussed in sections 7,8,9.
In section 10, we describe a deterministic version of the algorithm and prove that it works for a typical instance.
Finally, we coded the algorithm in C++, and also coded an algorithm for randomly sampling instances from the planted assignment distribution. We tested the algorithm on both planted assignment instances and other benchmarks from SATLIB (an on-line available library of SAT solvers and benchmarks for testing their quality). In section 12 we discuss the tests results.

## 5 Properties of a Random Instance

Throughout the discussion in this section, whenever we refer to an instance $I$, we mean an instance $I$ belonging to $\mathbb{P}_{d,n}$.

### 5.1 The Majority Vote

**Claim 5.1.** *Let $X$ be a random variable counting the number of variables in $I$ for which the majority vote disagrees with the original planted assignment. Then, $Pr[X > 2^{-d/20}n] \leq e^{-\Theta(n)}$*

We will prove the claim by first calculating the expectation of $X$, and then prove concentration around the expectation.

For a random variable $X$, we will use the following notation $X \sim B(p,n)$, to note that $X$ has binomial distribution over $n$ trials with success probability $p$.

**Lemma 5.1.** *Consider two binomial random variables, $X_1 \sim B(p,k_1)$ and $X_0 \sim B(p,k_0)$ s.t. $3k_0 > k_1 \geq k_0 > 0$. Then, $Pr[X_1 < X_0] \leq 2 \cdot e^{-p\frac{(k_1-k_0)^2}{2k_1}}$*

**Proof.**

$Pr[X_0 \geq X_1] \leq Pr[X_0 \geq a] + Pr[X_1 \leq a]$ for some $a$.

Take $a = p\left(\frac{k_1+k_0}{2}\right)$. The proof follows quite easily from the following Chernoff bound (for a binomial random variable $X$ with mean $\mu$):

$Pr[X \geq (1+\delta)\mu] \leq e^{\frac{-\delta^2\mu}{3}}, Pr[X \leq (1-\delta)\mu] \leq e^{\frac{-\delta^2\mu}{2}}, \delta \in (0,1)$

Set $\delta_0 = \dfrac{k_1-k_0}{2k_0}, \delta_1 = \dfrac{k_1-k_0}{2k_1}, \mu_i = p \cdot k_i, i = 0,1$

We can use the Chernoff bound in this from since $\delta_i < 1$ (from the conditions of the lemma).

$Pr[X_0 \geq k\dfrac{p_1+p_0}{2}] = Pr[X_0 \geq (1+\delta_0)\mu_0] \leq e^{-\left(\frac{k_1-k_0}{2k_0}\right)^2 \cdot pk_0} = e^{-\frac{p(k_1-k_0)^2}{2k_0}}$.

In the same manner, $Pr[X_1 \leq k\dfrac{p_1+p_0}{2}] \leq e^{-\left(\frac{k_1-k_0}{2k_1}\right)^2 \cdot pk_1} = e^{-\frac{p(k_1-k_0)^2}{2k_1}}$.

We conclude that, $Pr[X_0 \geq X_1] \leq e^{\frac{p(k_1-k_0)^2}{2k_1}} + e^{\frac{p(k_1-k_0)^2}{2k_0}} \leq 2e^{-\frac{p(k_1-k_0)^2}{2k_1}}$.

■

**Claim 5.2.** *For a variable $x$, the probability of the majority vote disagreeing with the planted assignment, is at most $e^{-d/19}$.*

**Proof.** Fix the variable $x$. Suppose w.l.o.g that $\varphi(x) = T$. We have $\binom{n-1}{2}2^2 \approx 2n^2$ clauses in which $x$ appears positively (call it a positive clause) and $\binom{n-1}{2}(2^2 - 1) \approx \frac{3}{2}n^2$ in which it appears negatively (a negative clause). The majority vote for $x$ disagrees with $\varphi$ if more negative-clauses were picked than positive ones. By plugging $k_1 = 2n^2, k_0 = \frac{3}{2}n^2, p = \frac{d}{n^2}$ in the previous lemma, we have that the probability of the majority vote disagreeing with $\varphi$ is at most

$$2e^{-\frac{(2n^2-1.5n^2)^2 d}{4n^2n^2}} = 2e^{-d/16} < e^{-d/19} \text{ for large enough } d$$

■

We are now ready to prove our main claim.

**Proof.** (Claim 5.1) We denote by $size(I)$ the number of clauses in $I$. Recall that $X$ counts the number of variables for which the majority vote disagrees with the original assignment.

$$\begin{aligned} Pr[X > 2^{-d/20}n] =& Pr[X > 2^{-d/20}n | size(I) \leq 2dn] Pr[size(I) \leq 2dn] + \\ & Pr[X > 2^{-d/20}n | size(I) > 2dn] Pr[size(I) > 2dn] \end{aligned}$$

First we will bound $Pr[X > 2^{-d/20}n|size(I) > 2dn]Pr[size(I) > 2dn]$.
The expected size of $I$ is $7dn/6$. $Pr[size(I) > 2dn] = Pr[size(I) > (1 + \frac{5}{7})\frac{7dn}{6}] < e^{-n/5}$ (due to the Chernoff bound).
$Pr[X > 2^{-d/20}n|size(I) > 2dn]Pr[size(I) > 2dn] \leq Pr[size(I) > 2dn] < e^{-n/5}$.
Define a new r.v. $Y = X|\{size(I) \leq 2dn\}$.
We will bound $Pr[X > 2^{-d/20}n|size(I) \leq 2dn] = Pr[Y > 2^{-d/20}n]$.
From the linearity of expectation, we get $E[X] \leq e^{-d/19}n$.

$$\begin{aligned} &E[X] = E[X|size(I) \leq 2dn]Pr[size(I) \leq 2dn] + E[X|size(I) > 2dn]Pr[size(I) > 2dn]. \\ &E[Y] = E[X|size(I) \leq 2dn] = (E[X] - E[X|size(I) > 2dn]Pr[size(I) > 2dn])\,/Pr[size(I) \leq 2dn] \\ &\leq E[X]/Pr[size(I) \leq 2dn] < e^{-d/19}n(1 - e^{-n/5})^{-1} \leq e^{-d/20}n \text{ for large enough } n \end{aligned}$$

We will prove concentration around the expectation of $Y$ using the method of martingales.
Define the following sequence of events $\{Z_i\}_{i=1}^{2dn}$, $Z_i$ stands for adding the $i$'$th$ clause.
Let $Y_i = E[Y|Z_1, Z_2, ..., Z_i]$. $Y_0 = E[Y] \leq e^{-d/20}n$, $Y_{2dn} = E[Y|Z_1, Z_2, ..., Z_{2dn}] = Y$.
The martingale difference $c_i = |Y_i - Y_{i-1}| \leq 3$, since every clause can ruin the majority vote for at most 3 variables.
Recall Azuma's inequality: $Pr[|Y_{2dn} - Y_0| \geq t] \leq e^{-t^2/2\sum_{i=1}^{2dn} c_i^2}$.
Plugging in the values, and setting $t = E[Y]$ we get that

$$Pr[Y \geq 2E[Y]] \leq e^{-(e^{-d/20}n)^2/2\cdot 2dn\cdot 9} \leq e^{-n/e^{d/5}} \text{ for large enough } d$$

We proved that w.p at least $1 - e^{-n/e^{d/5}}$, $Y < 2e^{-d/20}n < 2^{-d/20}n$ for large enough $d$.
$Pr[X > 2^{-\frac{d}{20}}n|size(I) \leq 2dn]Pr[size(I) \leq 2dn] \leq Pr[X > 2^{-\frac{d}{20}}n|size(I) \leq 2dn] \leq e^{-n/e^{d/5}}$.
For conclusion,

$$Pr[X > 2^{-\frac{d}{20}}n] \leq e^{-n/5} + e^{-n/e^{d/5}}$$

■

## 5.2 The Core Variables

The motivation for defining the core variables lies in the fact that those variables are easily set by the algorithm according to the original satisfying assignment. We now rigourously define a core group. We then show a procedure that finds w.h.p a specific core of $I$ w.r.t the majority vote assignment.

**Definition 5.1.** *(Core group) Let $\alpha < 1$ be a constant independent of $d$. We say that a set of variables $C$ is a* core *of $I$, w.r.t to an assignment $\psi$, if the following holds:*

- *$|C| \geq (1 - 3 \cdot 2^{-\alpha d})n$.*
- *Every variable in $C$ supports at least $\frac{d}{2} - 2\epsilon d$ clauses in which the other two variables are also in $C$.*

**Claim 5.3.** *Let $I$ be a random instance. Let $\varphi$ be the planted satisfying assignment of $I$. Let $\psi$ be an arbitrary assignment whose hamming distance from $\varphi$ is less than $3 \cdot 2^{-\alpha d}n$. Let $C$ be a core group of $I$ w.r.t to $\psi$. Then w.h.p, over the choice of $I$, for every $x \in C$, $\psi(x) = \varphi(x)$.*

To prove the claim we will need the following basic lemma. This lemma will serve us in other proofs as well, and is a basic ingredient in the correctness of the unassignment phase of the algorithm.
**Lemma 5.2.** *(density property) Let $c \gneqq 1$ be some constant. Let $I \in_R \mathbb{P}_{d,n}$. Then w.h.p (at least $1 - O(n^{2-2c})$), there exists no subset of variables $U$, s.t. $|U| \leq 3 \cdot 2^{-\alpha d}n$ and there are at least $c|U|$ clauses in $I$ containing two variables from $U$.*

*Remark 1:* Throughout our analysis, whenever we use Lemma 5.2 it holds that $c \geq \epsilon d/3$. In this case, and when $d$ is sufficiently large, the density property as stated in Lemma 5.2 fails to hold with probability at most $n^{-\Theta(d)}$.
*Remark 2:* The proof is basically the same as in [21]. We give it here for completeness and add a few more explanations. The last part of the proof is quite technical; the reader is referred to the appendix for the complete proof of the lemma.
**Proof.** Let $c \gneqq 1$ be some constant. For a fixed set $U$ with $|U| = k$, the number of clauses containing two variables from $U$ is

$$\binom{k}{2}(n-2)2^3 \leq 4k^2n$$

Each of the clauses is included independently w.p. $\frac{d}{n^2}$. Thus, the probability that $ck$ of them are included is at most

$$\binom{4k^2n}{ck}\left(\frac{d}{n^2}\right)^{ck} \leq \left(\frac{4k^2ne}{ck}\cdot\frac{d}{n^2}\right)^{ck} \leq \left(\frac{12kd}{cn}\right)^{ck}$$

We complete the proof by a union bound. The probability that any set $U$ has this property is at most

$$\sum_{k=2}^{3\cdot 2^{-\alpha d}n}\binom{n}{k}\left(\frac{12kd}{cn}\right)^{ck} = O(n^{2-2c})$$

■

**Proof.**(Claim 5.3) By contradiction, Let $U \subseteq C$ be the set of variables whose assignment under $\psi$ disagrees with $\varphi$. Every $x \in U$ supports at least $\frac{d}{2} - 2\epsilon d$ clauses in which the other two variables are also in $C$. $\psi(x) \neq \varphi(x)$, and since all the clauses $x$ supports are satisfied by $\varphi$, every such clause must contain another variable $x' \in C$ for which $\psi(x') \neq \varphi(x')$ (in particular, $x' \in U$).
Since the supporter of a clause is unique, we counted every clause (containing wrongly assigned variables) once.
To sum up, we have $|U|(\frac{d}{2} - 2\epsilon d)$ clauses constraining 2 variables from $U$. Since $\psi$ differs from $\varphi$ in at most $3 \cdot 2^{-\alpha d}n$ places, we know that $|U| \leq 3 \cdot 2^{-\alpha d}n$. However, $\frac{d}{2} - 2\epsilon d \geq 2$ for small enough $\epsilon$, and we have a contradiction to Lemma 5.2. ■

We now describe a procedure that w.h.p finds a core group of $I$ w.r.t the majority vote assignment.
*Let $V$ denote the set of all variables.*
*Let $W$ be the set of variables whose majority vote disagrees with $\varphi$.*
*Let $B$ be the set of variables which are not strong supporters w.r.t. $\varphi$.*
*Let $H \subset V$ be constructed as follows*

1. *Let $H_0 = V \setminus (B \cup W)$.*

2. *While there is a variable $a_i \in H_i$ that supports less than $\frac{d}{2} - 2\epsilon d$ clauses in which the other two variables are also in $H_i$, define $H_{i+1} = H_i \setminus \{a_i\}$.*

3. *Let $a_m$ be the last variable removed at step 2. Define $H = H_{m+1}$.*

**Claim 5.4.** *W.h.p, $H$ is a core group of $I$ w.r.t to the majority vote.*

**Proof.** Let $\psi$ be the assignment induced by the majority vote. To prove that $H$ is a core group w.r.t $\psi$ we need to prove 2 properties.
First we show that every variable in $H$ supports at least $\frac{d}{2} - 2\epsilon d$ clauses (w.r.t $\psi$) in which the other two variables are also in $H$.
By the construction of $H$, every variable in $H$ supports at least $\frac{d}{2} - 2\epsilon d$ clauses, w.r.t $\varphi$, in which the other two variables are also in $H$. However, the first stage of constructing $H$ was disqualifying

all variables whose assignment under $\psi$ differs from $\varphi$. Thus we get this property w.r.t to $\psi$ as well. Next, we need to show that $|H| \geq (1-3\cdot 2^{-\alpha d})n$. The next claim gives us this property and finishes the proof. ∎

**Claim 5.5.** *W.h.p.* $|H| \geq (1 - 3 \cdot 2^{-\alpha d})n$.

In order to prove the claim, we need the following lemma.
**Lemma 5.3.** *There are at least* $(1 - 2^{-\alpha d})n$ *variables in* $I$*, which are strong supporters w.r.t* $\varphi$.

**Proof.** Every variable $x$ is expected to support $\binom{n-1}{2} \cdot \frac{d}{n^2} = \frac{d}{2} + O(\frac{1}{n})$ clauses.
Recall that a variable $x$ is called a strong supporter if it supports at least $\frac{d}{2} - \epsilon d$ clauses.
By applying the Chernoff bound (which is applicable since all the clauses are chosen independently), we have that the probability that a variable is not a strong supporter is at most $e^{-\epsilon^2 d}$. We call such variable *bad*.
Let $X$ be a r.v. counting the number of *bad* variables. $E[X] = e^{-\epsilon^2 d}n$.
We follow the same pattern as in Claim 5.1. We define a clause-exposure martingale, and prove concentration around the expectation using Azuma's inequality (we can follow this pattern since the martingale difference remains 3). Namely, with probability $1 - e^{-\Theta(n)}$, at least $2^{-\epsilon^2 d}n$ variables support about the right number of clauses.
Let $\alpha = \max(\frac{1}{20}, \epsilon^2)$. Since $\epsilon$ is independent of $d$, so is $\alpha$. ∎

We are now ready to prove Claim 5.5.
**Proof.**(Claim 5.5) By Claim 5.1 and Lemma 5.3, we know that w.h.p $|B \cup W| \leq 2 \cdot 2^{-\alpha d}n$.
Let $m_0 = 2^{-\alpha d}n$, Suppose by contradiction that $m \geq m_0$.
Let $U = B \cup W \cup \{a_1, a_2, ..., a_{m_0}\}$.
Every $a_i$ supports at least $\frac{d}{2} - \epsilon d$ clauses, but it supports at most $\frac{d}{2} - 2\epsilon d$ clauses with only variables of $H_i$. Hence, every $a_i, i \leq m_0$ supports (and in particular appears in) at least $\epsilon d$ clauses with another variable of $B \cup W \cup \{a_1, a_2, ..., a_{i-1}\} \subseteq U$.
Note that all clauses are distinct since every clause has a unique supporter (from the definition of support).
$|U| \leq 2 \cdot 2^{-\alpha d}n + m_0, m_0 = 2^{-\alpha d}n \Rightarrow m_0 \geq \frac{|U|}{3}$.
Hence, we have $\epsilon d m_0 \geq \frac{\epsilon d |U|}{3}$ clauses that contain two variables from $U$. $|U| \leq 3 \cdot 2^{-\alpha d}n$, $\frac{\epsilon d}{3} \geq 2$ for large enough $d$. This contradicts Lemma 5.2.
To sum up, w.h.p we eliminated at most $3 \cdot 2^{-\alpha d}n$ variables. The claim follows. ∎

**Claim 5.6.** *For every variable in* $H$*, w.h.p its majority vote agrees with* $\varphi$.

**Proof.** The claim follows immediately from combining the following 3 facts that hold w.h.p for $I$. The majority vote assignment differs from $\varphi$ for at most $2^{-\alpha d}n$ variables (Claim 5.1), $H$ is a core group w.r.t the majority vote (Claim 5.4) and every core group is set correctly (Claim 5.3).
Note that in our specific core, the claim follows immediately from the construction of $H$ (the first stage was removing all variables whose majority vote differs from $\varphi$). ∎

Note that by combining Lemma 5.2 and the definition of a core, it follows immediately that the core group is of size at least $3 \cdot 2^{-\alpha d} n$. However, we still haven't used the requirement that the core group contains almost all variables (and not just some linear amount of variables, e.g. $3 \cdot 2^{-\alpha d} n$ variables). We need the latter for proving that the unassignment phase does not leave wrongly assigned variables.

For the rest of the paper, when we say *core group* we refer to the specific $H$ we defined above. We do not use the general notion of core since in the next section we use the specific structure of $H$.

## 5.3 The Connected Components

**Definition 5.2.** *Let $G(U, E)$ be the graph defined as follows. $U$ is the set of variables not in the core ($U = V \setminus H$). $(x_i, x_j) \in E$ if there exists some clause containing both $x_i, x_j$.*
**Claim 5.7.** *If $|U| < 3 \cdot 2^{-\alpha d} n$, the largest connected component in $G(U, E)$ is of size $\leq \log n$ w.h.p.*

*Remark:* a similar claim is proved in [21]. However, for completeness and extra details, we give here a self contained proof of the claim without appealing to [21].
**Proof.** Our strategy is to show that w.h.p, the largest tree in the graph is of size $\log n$.
Let $T$ be a fixed tree on $\log n$ variables, and let $T'$ be a fixed collection of clauses such that each edge of $T$ is induced by some clause of $T'$. Let $V(T)$ be the set of vertices of $T$, and $E(T)$ be the set of edges of $T$.
We say that an edge of $T$ is *covered* by $T'$ if there exists a clause in $T'$ that contains both its endpoints.
We say that $T'$ is *minimal* if by deleting a clause from $T'$, $T$ is not covered by $T'$ anymore.
From the definition we have that every clause in $T'$ has an edge in $T$ that it covers uniquely (otherwise, remove that clause and be left with a smaller set that still covers $T$ in contradiction to the minimality of $T'$), hence $|T'| \leq |E(T)| = |V(T)| - 1$ since $T$ is a tree.

Let $I$ be the $3CNF$ instance, our goal is to show that $Pr[T' \subseteq I \text{ and } V(T) \cap H = \emptyset]$ is small enough, such that by taking the union bound on all possible trees and possible extensions to minimal spanning sets, we will get probability $o(1)$ of having a tree of that size.
Both $Pr[T' \subseteq I]$ and $Pr[V(T) \cap H = \emptyset]$ are easily calculated (for a fixed $T, T'$). However, since $H$ is not a random subset of $V$ (and depends on the structure of $I$), it need not follow that $Pr[T' \subseteq I \text{ and } V(T) \cap H = \emptyset] \leq Pr[T' \subseteq I] \cdot Pr[V(T) \cap H = \emptyset]$.
We need a more delicate analysis. Instead of looking at $H$, we will look at another set $H'$ which is built similarly to $H$, but also depends on $T$. The intuition for defining $H'$ is deferred to the discussion in Claim 5.10.

Let $J \subseteq V(T)$ be a set of variables which appear in at most 6 clauses of $T'$.
For a set of clauses $I$, let $|I|$ denote the number of clauses in $I$.

**Claim 5.8.** $|J| \geq \frac{|V(T)|}{2}$ *(same as Claim 6 at [21])*

**Proof.** Suppose that $|J| < \frac{|V(T)|}{2}$, then more than $\frac{|V(T)|}{2}$ variables of $T'$ appear in more than 6 clauses. All in all they appear in at least $6 \cdot \frac{|V(T)|}{2}$ clauses. How many clauses have we counted more than once? every clause was counted at most 3 times, once for every variable appearing in it. Hence we have at least $\frac{1}{3} \cdot 3 \cdot |V(T)|$ distinct clauses in $T' \Rightarrow |T'| \geq |V(T)|$.
On the other hand, we have that $|T'| \leq |V(T)| - 1$. We have reached a contradiction regarding the size of $T$. ■

We define the set of variables $H'$ by the following iterative procedure:
*Let $V$ denote the set of all variables.*
*Let $W$ be the set of variables for which the majority vote is wrong or for which the majority is right with advantage smaller than 7.*
*Let $B$ be the set of variables which are not strong supporters w.r.t. $\varphi$.*
*Let $H \subset V$ be constructed as follows:*

1. *Let $H'_0 = V \setminus \{W \cup B \cup (V(T') \setminus J)\}$.*
2. *While there is a variable $a_i \in H'_i$ that supports less than $\frac{d}{2} - 2\epsilon d$ clauses with only variables of $H'_i$, define $H'_{i+1} = H'_i \setminus \{a_i\}$.*
3. *Let $a_m$ be the last variable removed at step 2. Define $H' = H'_{m+1}$.*

The difference between the procedure for $H$ and $H'$ is in the definition of $W$ and in the definition of $H'_0$.

**Claim 5.9.** $|H'| \geq (1 - 2^{-\Theta(d)})n$

**Proof.** The proof is essentially the same as Claim 5.5, we just need to be careful about two things. The size of $V(T') \setminus J$ and the size of $W$.
We know that $|V(T') \setminus J| \leq |V(T')| \leq 3|V(T)| \leq 3 \cdot \log n$.
We proved in section 5.1, that the majority is correct for $(1-2^{\frac{-d}{20}})$-fraction of the variables. Suppose a variable $x$ was originally assigned $T$; $x$ is expected to appear positively $2d$ times and negatively $1.5d$ times. The expected gap is then $\frac{d}{2}$. We demand a gap of at least 7, which is much smaller than $\frac{d}{2}$. Hence we can repeat section 5.1, and prove the same result when demanding a gap of 7 in the majority.
From those observations it is clear that we can repeat exactly the same proof as in Claim 5.5 and get the same result regarding the size of $H'$. ■

**Claim 5.10.** *$Pr[T' \subset I$ and $V(T) \cap H = \emptyset] \leq Pr[T' \subset I]Pr[J \cap H' = \emptyset]$.*

The key for proving the claim is in the following inclusion.

**Lemma 5.4.** *Let $F$ be a set of clauses and let $H(F \cup T')$ be the value of $H$ if $I = F \cup T'$. Let $H'(F)$ be the value of $H'$ if $I = F$. Then $H'(F) \subseteq H(F \cup T')$.*

This lemma clarifies the motivation for defining $H'$. It is not true that $H(F) \subseteq H(F \cup T')$ for all $F$. This is because a variable which appeared in $H(F)$ could disappear from $H(F \cup T')$ since the clauses of $T'$ would make the majority vote wrong for it (however, they cannot reduce its support). Same considerations are true in [14],[5],[21]; In [5] the set of edges in $T'$ might cause a vertex to have more neighbors than expected and hence be excluded from $H(F \cup T')$. We overcome this problem by deleting from $H'$ vertices with high degree in $T'$ (in our case, vertices with degree greater than 6). This on the one hand reduces the effect of $T'$ on $H'$, but on the other leaves enough vertices in $J$ (at least $\frac{|V(T)|}{2}$) such that the value $Pr[J \cap H' = \emptyset]$ would be sufficiently small to help bound the probability for a tree.

The proofs of Claim 5.10 and Lemma 5.4 are quite technical and might distract the reader from the main ideas of this section. The proofs are omitted here and are given fully in the appendix.

**Lemma 5.5.** *Let $\beta \in (0, 1]$,$k = o(n)$. Then, $\frac{\binom{\beta n}{k}}{\binom{n}{k}} \leq \beta^k$*

**Proof.** We recall Stirling's approximation $n! \approx n^n e^{-n}\sqrt{2\pi n}$.
When we have binomial coefficients the $e^{-n}$ factor cancels out.

$$\frac{\binom{\beta n}{k}}{\binom{n}{k}} = \frac{(\beta n)!}{k!(\beta n - k)!} \cdot \frac{k!(n-k)!}{n!} \approx \frac{(\beta n)^{\beta n}\sqrt{2\pi \beta n}}{(\beta n - k)^{\beta n - k}\sqrt{2\pi(\beta n - k)}} \cdot \frac{(n-k)^{n-k}\sqrt{2\pi(n-k)}}{n^n\sqrt{2\pi n}} =$$
$$\left(\frac{\beta n}{\beta n - k}\right)^{\beta n} \left(\frac{\beta n - k}{n - k}\right)^{k} \left(\frac{n-k}{n}\right)^{n} \sqrt{\frac{\beta n - \beta k}{\beta n - k}} \approx 1^{\beta n}\beta^k e^{-k}\sqrt{1} \leq \beta^k$$

The last $\approx$ is the asymptotic behavior when $n$ is sufficiently large. This is true because $k = o(n)$. ■

**Lemma 5.6.** $Pr[J \cap H' = \emptyset] \leq n^{-\Theta(d)}$

**Proof.** Since the choice of $H'$ is independent of $J$ we have that

$$Pr[J \cap H' = \emptyset] \leq \frac{\binom{n-|J|}{|H'|}}{\binom{n}{|H'|}} = \frac{\binom{n-|H'|}{|J|}}{\binom{n}{|J|}} = \frac{\binom{e^{-\Theta(d)}n}{\log n/2}}{\binom{n}{\log n/2}} \leq \left(e^{-\Theta(d)}\right)^{\log n/2} = n^{-\Theta(d)}$$

■

**Corollary 5.11.** $Pr[T' \subset I \wedge V(T) \cap H = \emptyset] \leq n^{-\Theta(d)}(d/n^2)^{|T'|}$

**Proof.** By Claim 5.10 we have that $Pr[T' \subset I \wedge V(T) \cap H = \emptyset] \leq Pr[T' \subset I]Pr[J \cap H' = \emptyset]$. By the previous lemma we have that $Pr[J \cap H' = \emptyset] \leq n^{-\Theta(d)}$. Since the clauses of $I$ are chosen independently, we have that $Pr[T' \subset I] = (d/n^2)^{|T'|}$.
All in all, $Pr[T' \subset I \wedge V(T) \cap H = \emptyset] \leq n^{-\Theta(d)}(d/n^2)^{|T'|}$ ■

**Lemma 5.7.** *Let $T$ be a fixed tree with $k$ vertices. The number of minimal sets of clauses that span $T$ is at most*

$$\sum_{s=0}^{k/2} \binom{k}{2} 7^{k-s-1} n^{k-2s-1}$$

**Proof.** Let $T'$ be a minimal spanning set. We proved that a minimal set contains at most $k-1$ clauses. Every clause may cover up to 2 edges (otherwise we would have a cycle in $T$). We can extend a tree to a minimal set by having $s$ clauses cover 2 edges of $T'$ and $k-1-2s$ clauses cover one edge. The total number of clauses in $T'$ is $k-s-1$. For every clause we have $(2^3-1)$ ways to set the signs of the variables. For every clause that covers one edge, we have $(n-2)$ ways to choose the third variable. A clause that covers 2 edges is determined uniquely by the two edges. We have at most $\binom{k}{2}$ ways to form the clauses that cover 2 edges. $s$ may range from 0 to $k/2$. ∎

**Theorem 5.12.** *(Cayley) The number of spanning trees of $K_n$ is $n^{n-2}$*

Four different proofs are given at [3].

**Claim 5.13.** *The probability of a tree of size $\log n$ in the induced graph is $o(1)$.*

**Proof.** We prove by a union bound.

$$\sum_{s=0}^{\log n/2} \underbrace{\binom{n}{\log n} \cdot (\log n)^{\log n - 2}}_{\text{number of trees of size } \log n} \cdot \underbrace{7^{\log n - s - 1} \cdot n^{\log n - 2s - 1} \binom{\log n}{2}}_{\text{extend to a minimal set}} \underbrace{\cdot (d/n^2)^{\log n - s - 1} \cdot n^{-\Theta(d)}}_{\text{variables don't intersect } H \text{ (cor. 5.11)}} \leq$$

$$\sum_{s=0}^{\log n/2} (\frac{e \cdot n}{\log n})^{\log n} \cdot (\log n)^{\log n} \cdot n^{\log 7 + \log d} \cdot n^{\log n - 2s - 1 - 2(\log n - s - 1)} \cdot n^{-\Theta(d)} =$$

$$\sum_{s=0}^{\log n/2} n^{-\Theta(d) + \log 7 + \log d + 1} \cdot n^{\log n} \cdot n^{-\log n + 1} = \sum_{s=0}^{\log n/2} n^{-\Theta(d) + log 7 + \log d + 2}$$

$$= n^{-\Theta(d)} = o(1)$$

∎

Claim 5.7 follows easily from the last claim. ∎

## 5.4 The Random Walk

**Claim 5.14.** *Consider a satisfiable* $3CNF$ *formula over* $n$ *variables, whose largest connected component is of size at most* $\log n$. *Then, using at most* $3tn$ *steps, the random walk procedure finds a satisfying assignment w.p.* $> 1 - \frac{n}{\log n} e^{-t/n}$.

Let $\varphi$ be a satisfying assignment. First consider a random walk within some connected component. Let $s \leq \log n$ be the number of variables in the component.
We start with a random assignment to the variables of the component. The assignment is of Hamming distance $d$ from the $\varphi$, for some $0 \leq d \leq s$. In every step, we choose an unsatisfied clause and flip a randomly picked variable in it (see [32]). After every flip, either we increased the Hamming distance from $\varphi$ or decreased it (happens w.p. at least $\frac{1}{3}$). Thus, one can view the random walk algorithm as if making a random walk on a line of length $s$. Every node on that line represents the hamming distance from $\varphi$. Our goal is to reach node 0 (hamming distance 0).
We will cite 2 lemmas whose proof is given fully at [32]. The two lemmas refer to a random walk on a line (as described above).
**Lemma 5.8.** *Let* $d \leq s$ *be the initial hamming distance from the original satisfying assignment. Let* $q_d$ *be the probability that we reached node* 0 *using* $3s$ *steps. Then,*

$$q_d \geq \left(\frac{1}{2}\right)^d$$

**Lemma 5.9.** *Let* $q$ *be the probability that the random walk, starting from a* random initial assignment*, would reach node* 0 *within* $3s$ *steps. Then,*

$$q \geq \left(\frac{3}{4}\right)^s$$

The following two discussions (random walk with/without restart) comprise the proof of Claim 5.14.

### 5.4.1 Random Walk With Restart

First we will consider a random walk where we restart the procedure if no satisfying assignment was reached after $3s$ steps.
In every try, we have probability at least $q$ to succeed. Suppose we restart the procedure for $t$ times. Then the probability that we failed in all $t$ times is at most $(1-q)^t \leq e^{-qt}$. By plugging in the value of $s = \log n$, and $q = n^{-\log \frac{4}{3}}$ (by Lemma 5.9), we get that after at most $t \cdot 3 \log n$ steps, w.p. at least $1 - e^{-t/n^{\log \frac{4}{3}}}$ we find a satisfying assignment.
W.l.o.g assume that the connected components are of size exactly $\log n$.
We run this procedure separately in every connected component. Since we have at most $\frac{n}{\log n}$ connected components, using $t \cdot 3 \log n \cdot \frac{n}{\log n} = 3tn$ flips, w.p. at least $1 - \frac{n}{\log n} e^{-t/n^{\log \frac{4}{3}}}$ we find the satisfying assignment in all components (union bound).

### 5.4.2 Random Walk Without Restart

We can think of the procedure within the connected component as follows. We start with a random assignment. Suppose the worst case - the distance from the original assignment is $s$. After $3s$ steps, we "restart" the procedure. Our restart procedure is empty, since we again assume that we reached the worst case after $3s$ steps. We now apply the same analysis as above. This time however, $q = (\frac{1}{2})^s = n^{-1}$ (by Lemma 5.8). We get that after at most $3t \log n$ steps, w.p. at least $1 - e^{-t/n}$ we find a satisfying assignment.

We now run a random walk on the whole formula, and bound the total number of steps by $3tn$. We tick the time in every connected component (separately) when we flip a variable belonging to it. With probability at least $1 - e^{-t/n}$, we will find a satisfying assignment after $3t \log n$ steps/ticks in every component (and then never visit it again). Since we have at most $\frac{n}{\log n}$ connected components, using $(3t \log n) \cdot \frac{n}{\log n} = 3tn$ steps, w.p. at least $1 - \frac{n}{\log n} e^{-t/n}$ we will find the satisfying assignment in all of them.

## 5.5 Probability for a Typical Instance

Define the following events:

$A_1 =$ the majority vote is correct for more than $(1 - 2^{-\frac{d}{20}})$-fraction of the variables.

$A_2 =$ Lemma 5.3 and Lemma 5.2 hold (given $A_1$, this implies that the core group is of desired size).

$A_3 =$ The graph induced on non-core variables breaks into connected components of size at most $\log n$.

$$Pr[I \text{ is typical}] \geq Pr[\bigwedge_{i=1}^{3} A_i] = Pr[A_3|A_1, A_2] Pr[A_2|A_1] Pr[A_1].$$

$$Pr[A_1] \geq 1 - e^{-\Theta(n)},\ Pr[A_2|A_1] \geq Pr[A_2] = 1 - n^{-\Theta(d)},\ Pr[A_3|A_1, A_2] \geq 1 - n^{-\Theta(d)}.$$

To sum up, the probability of an instance having all properties behaves like $1 - n^{-\Theta(d)}$.

# 6 Algorithm Analysis on a Typical Instance

We now get back to the algorithm we described in section 3. In this section we prove that if $I$ is a typical instance (namely, all properties described in the previous section hold for $I$), then the algorithm finds a satisfying assignment for $I$ w.p at least $1 - \frac{n}{\log n} e^{-t/n}$ (the failure probability is that of the random walk procedure).

Note that the algorithm does not know $d$ nor $n$. Let us first deal with the issue of not knowing $n$. Using the Chernoff bound, we expect at most $2^{-\Theta(d)} n$ variables not to appear at all in $I$. Let $n'$ be the number of variables actually appearing in $I$. Using same concentration arguments as in section 5.1, w.h.p $n' \geq n(1 - 2^{-\Theta(d)})$. The algorithm doesn't use $n$ explicitly, rather it uses $\log n$ (for the stopping condition). $\log n' \geq \log n + \log(1 - 2^{-\Theta(d)}) \Rightarrow \log n - \log n' < \frac{1}{2}$ for large enough $d$. Thus, we can safely take $\log n' + 1$ as an upper bound on $\log n$.
Obviously, $n'$ can be calculated from $I$ by simply counting the number of variables appearing in $I$. Moving to the issue of not knowing $d$, we can approximate $d$ from the clause to variable ratio in $I$ (the expected ratio is $7d/6$). However, we choose to overcome the problem of not knowing $d$ by running the algorithm with $d' = 1 ... c_0 \log n$ (recall that $p = c_0 \log n / n^2$ is the threshold probability above which the majority vote sets all variables correctly w.h.p). If $d \geq c_0 \log n$, then the majority vote set all the variables correct w.h.p and we don't need to know $d$ (since we will not reach the unassignment phase). If $d < c_0 \log n$, then in one of the iterations we will run with the correct $d'$. The motivation for choosing this solution will become clear when discussing the semi-random model.

Another problem is that $d$ need not take integral values. We run only for integral values of $d'$. However, the unassignment is done according to $\frac{d'}{2} - 2\epsilon d'$. We can assume w.l.o.g that $d$ is integral since $\epsilon$ is a fraction anyway.

In this section, we analyze the iteration of the algorithm in which $d' = d$. Thus, we abuse the notation and use $d$ for both $\mathbb{P}_{d,n}$ and the parametrization of the algorithm ($d'$).

## 6.1 The Unassignment Phase

Let us first recall the unassignment phase. After we assign all variables according to their majority vote, we unassign every variable who is not even a mild supporter (namely, it supports less than $\frac{d}{2} - 2\epsilon d$ clauses) w.r.t the current assignment. We keep going until no further unassignment is possible. Recall also that we count a clause in the support only if all its three variables are assigned.

**Claim 6.1.** *After the unassignment phase ends, at least $(1 - 3 \cdot 2^{-\alpha d})n$ of the variables remain assigned, and every assigned variables is assigned correctly.*

Before proving the claim we give some intuition for the correctness of the unassignment phase.

If a variable is set incorrectly, and survives the unassignment phase, it means that it appears with many other wrongly assigned variables, which survived the unassignment phase as well (at least in all $d/2 - 2\epsilon d$ clauses it currently supports). Every one of them also supports many clauses (since it remained assigned), and thus also appears with many wrongly assigned variables, and so forth. This "avalanche effect" leads to the fact that w.h.p we have many wrongly assigned variables. However, this contradicts the properties of the majority vote.

**Claim 6.2.** *After the unassignment phase ends, all core variables remain assigned.*

**Proof.** As we saw earlier (Claim 5.4), $H$ is a core of $I$ w.r.t the majority vote assignment. By definition of core, every variable in $H$ supports at least $\frac{d}{2} - 2\epsilon d$ clauses with only variables of $H$ w.r.t the majority vote. Since we start the unassignment phase with a majority vote assignment, in the first iteration of unassignment, every variable of $H$ indeed supports at least $\frac{d}{2} - 2\epsilon d$ clauses. Hence no variable of $H$ is unassigned. Since no variable of $H$ is unassigned, every variable of $H$ passes the test in the second iteration as well. By induction one can easily show that all variables of $H$ remain assigned throughout the unassignment phase.

Note that in this proof we didn't use the structure or $H$, only the fact that $H$ is a core group. Hence, we actually proved that every core group survives the unassignment phase. ■

**Claim 6.3.** *After the unassignment phase ends, all assigned variables are assigned correctly.*

**Proof.** By the previous lemma, we know that all variables of $H$ are assigned, and by Claim 5.6 we know that they are assigned correctly.

Let $U$ be the set of variables which remain assigned after the unassignment phase ends, but whose assignment is wrong. Every variable which remains assigned, supports at least $\frac{d}{2} - 2\epsilon d$ clauses where all variables are assigned (since we count support only on totaly assigned clauses). Since the variable is wrongly assigned (its assignment should be false), and every clauses is satisfied by $\varphi$, we must have that in every clause that it supports, there is another variable which is wrongly assigned. Again, all clauses are distinct since every clause has a unique supporter.

Now we use the requirement regarding the size of the core. We have $|U| \cdot (\frac{d}{2} - 2\epsilon d)$ clauses containing two variables from $U$.

$U \cap H = \emptyset$ (since $H$ is assigned correctly), $|H| \geq 1 - 3 \cdot 2^{-\alpha d} n \Rightarrow |U| \leq 3 \cdot 2^{-\alpha d} n$. In addition $\frac{d}{2} - 2\epsilon d \geq 2$ for small enough $\epsilon$. We have a contradiction to Lemma 5.2.

Since the instance is typical, this rules out the existence of such $U$ and finishes the proof of the claim. ■

The proof of Claim 6.1 follows immediately from combining Claims 5.4 ($H$ is a core), 6.2 and 6.3.

## 6.2 The Connected Components

**Definition 6.1.** *Let $G'(U', E')$ be the graph defined as follows. $U'$ is the set of* unassigned *variables. $(x_i, x_j) \in E'$ if there exists some clause containing both $x_i, x_j$.*

**Claim 6.4.** *The largest connected component in $G'(U', E')$ is of size $\leq \log n$ w.h.p.*

**Proof.** Let $G(U, E)$ be the graph defined in subsection 5.3. Recall that $U$ is the set of variables not in the core. Claim 6.2 tells us that all variables of the core remain assigned, hence all unassigned variable are not in the core. In particular, $U' \subseteq U$. Hence, $G'$ is a subgraph of $G$. The claim follows immediately from this fact and Claim 5.7. ■

### 6.3 Unit Clause Propagation

**Definition 6.2.** *We say that the unit clause propagation reached a* contradiction*, if at some point, we have both $x$ and $\bar{x}$ as two unit clauses, for some variable $x$.*

It is easy to see that if a contradiction is reached, the algorithm will not find a satisfying assignment.

**Claim 6.5.** *If $I$ is satisfiable and typical, then the unit clause propagation does not reach a contradiction.*

**Proof.** The assignments given to variables in the unit clause propagation are the only possible ones for the formula to be satisfied. If we find a contradiction this means that the unassignment phase did not have the property that by the end, all variables remaining assigned are set correctly. However, for a typical instance this doesn't happen (Claim 6.3), hence contradiction is not reached. ■

**Corollary 6.6.** *All variables set by the unit clause propagation are correct.*

**Proof.** As mentioned above, the assignments given to variables are the only possible ones for the formula to be satisfied. Since the unassignment phase was correct, so must the unit clause propagation be. ■

### 6.4 The Random Walk Phase

**Claim 6.7.** *Consider the subformula induced by unassigned variables. Using at most $3tn$ steps, the random walk procedure finds a satisfying assignment for the subformula.*

**Proof.** First observe that the remaining subformula is satisfiable (since all assigned variables were set correctly, and $I$ is satisfiable). Our claim follows immediately by combining Claims 5.14 and 6.4 (here we can assume w.l.o.g that the subformula is in $3CNF$ form). ■

In order to use the restart mechanism in the random walk, we need to know exactly the connected components (in order to count the number of steps that we spent in every connected component). This can be achieved by running a BFS procedure on the whole formula, and getting the connected components.

### 6.5 Success Probability

We analyze the success probability of the algorithm on $I \in_R \mathbb{P}_{d,n}$ (namely, the probability is taken over the random coins in the choice of $I$).
Let $R$ be the event that the RWalkSat procedure, performed on the subformula induced by unassigned variables, converges to a satisfying assignment in polynomial time.

$$Pr[\text{The algorithm finds a satisfying assignment for } I] \underbrace{\geq}_{\textit{section 6}}$$
$$Pr[I \text{ is typical} \wedge R \text{ happens}] \underbrace{\geq}_{\textit{subsection 5.5 and Claim 6.7}} 1 - n^{-\Theta(d)} - \frac{n}{\log n} e^{-t/n}.$$

## 7 The Semi-Random Model

We now move to the semi-random model. First we will define the semi-random setting in which we are working. Next, we will show some adversarial strategies that fail Flaxman's algorithm. We then describe the properties that a semi-random instance has w.h.p (an define the notion of a typical semi-random instance). We finish by proving that our algorithm finds a satisfying assignment w.h.p for a typical semi-random instance.

### 7.1 Generating an Instance

The generation of a $3CNF$ instance can be viewed as taking place in two phases. The first phase is exactly what is described in section 2 (namely, generating a completely random instance from $\mathbb{P}_{d,n}$ with a planted assignment $\varphi$).
We say that a clause $c$ is *3-satisfiable* with respect to an assignment $\varphi$ if all three literals of $c$ are satisfied by $\psi$.
In the second phase, an adversary adds as many 3-satisfiable clauses (with respect to $\varphi$) as he wishes. The clauses are then shuffled (namely, for every clause, one cannot tell in which phase the clause was added).
Let $I$ be the final instance we got. We denote by $I_R$ the random part of $I$ and by $I_A$ the set of clauses which the adversary added. $I = I_R \cup I_A, I_A \cap I_R = \emptyset$.
We note that $I$ is satisfiable, since at least $\varphi$ satisfies it.

We can think of the adversary as planting "guiding tips" along the formula. The clauses that he adds are clauses which are "good" for many 3SAT solvers. E.g, for a random walk procedure, on such clause if unsatisfied, the random walk makes a correct move w.p. 1. For a majority vote procedure, every such clause pushes the majority in the correct direction. Thus, one might expect a natural 3SAT algorithm to do at least as well on such semi-random instances as on complectly random ones.

Throughput the analysis in the semi-random setting, we will point out whenever we used the fact the the adversary clauses are 3-satisfiable and point out what could have gone wrong unless they were 3-satisfiable.

### 7.2 Adversarial Strategies

For the analysis of the unassignment phase, both we and Flaxman use the fact that when starting the unassignment procedure the assignment at hand is of distance at most $2^{-\Theta(d)}n$ from the original satisfying assignment. We start the algorithm with a majority vote assignment, which w.h.p gives us the desired distance. Flaxman looks at a more general distribution for which the majority need not necessarily work. Hence, Flaxman begins the algorithm with a spectral approximation. He proves that the distance of the approximation from the original assignment is at most $O(n/d)$. As we said, Flaxman uses a tighter bound in the analysis of the unassignment phase. In order to achieve the tighter bound he has a refinement phase before starting the unassignment.

In Flaxman's analysis, the core variables are defined differently. The main difference is that a variable would not belong to the core if it appears in too many clauses or supports too many clauses. This definition may sound unnatural. Why should we care if a variable appears too many times or supports too many clauses. The reason for this definition is that in order to prove that the refinement phase leaves the core variables correct (a crucial point in the algorithm), Flaxman uses the bound on the number of appearances of a core variable. Flaxman uses the same condition for the core variables as an unassignment criteria (namely, unassign variables which appear in too many clauses or support to many clauses).

However, it is easy to see that unassigning according to the support count (same procedure we had, and adjusting the parameters to fit [21]) leaves Flaxman's core variables assigned and preserves the property that all remaining assigned variables are correct. In particular, there is no need to unassign a variable only because it appears in too many clauses or supports too many. The unassignment procedure of Flaxman works well in the random model. However, when moving to the semi-random setting, one needs a more delicate unassignment procedure (e.g., the one we have in our algorithm). We will present an adversarial strategy which demonstrates the need for a more delicate unassignment procedure.

Another problem that arises when moving to the semi-random model is in the last phase of Flaxman's algorithm - the exhaustive search. Since Flaxman's algorithm is deterministic (and so is ours, up to the random walk phase), the adversary can run the algorithm on the random instance, see the set of unassigned variables and add clauses that interconnect them to create a large connected component. We expect a linear (small, but still linear) fraction of the variables to be unassigned. The exhaustive search runs w.h.p in exponential time in the size of the connected

component (in our case exponential in $n$). Hence the exhaustive search fails w.h.p to find a satisfying assignment in polynomial time.
Note that adding clauses may change the set of unassigned variables. We will present an adversary that adds clauses s.t the set of unassigned variables does not decrease (namely, after adding clauses the new set of unassigned variables contains the old one).

We fixed this problem by replacing the exhaustive search with a random walk procedure. For the rest of this subsection we will assume, w.l.o.g, that the algorithm in [21] also starts with a majority vote instead of a spectral approximation. We can assume this because the bound, proven in [21], for the error of the spectral approximation, is weaker than the bound we proved using the majority vote. Furthermore, the analysis in [21] only uses the bound itself and not the fact that the approximation came from a spectral technique.

### 7.2.1 Preliminaries

Let $I$ be a completely random instance generated as described in section 2. The following discussion refers to such an instance.
Let $X \sim B(p, n)$ s.t $\mu = E[X] = np$ is constant.
Let $A_{X,b}$ be the event that $X \geq \mu + b$ ($b \geq 0$ is also constant).
**Claim 7.1.** *$Pr[A_{X,b}] \geq \omega$, where $\omega$ is some constant depending on $b, \mu$.*

To prove the claim we will first need the following lemma.
**Lemma 7.1.** $\binom{m}{k} \geq (\frac{m-k}{k})^k$

**Proof.**

$$\binom{m}{k} = \frac{m!}{(m-k)! \cdot k!} = \frac{m(m-1)\cdots(m-k+1)}{k!} \geq \left(\frac{m-k}{k}\right)^k$$

■

**Proof.**(Claim 7.1)

$$\begin{aligned} Pr[A_{X,b}] &\geq \binom{n}{\mu+b}\left(\frac{\mu}{n}\right)^{\mu+b}\left(1-\frac{\mu}{n}\right)^{n-(\mu+b)} \\ &\geq \binom{n}{\mu+b}\left(\frac{\mu}{n}\right)^{\mu+b}\left(1-\frac{\mu}{n}\right)^{n} \\ &\geq e^{-\mu}\cdot\left(\frac{n-(\mu+b)}{\mu+b}\cdot\frac{\mu}{n}\right)^{\mu+b} \geq \omega \end{aligned}$$

■

In a similar way one can prove constant probability for the event $X \leq \mu - b$ ($b \in [0, \mu]$).

**Claim 7.2.** *In both algorithms, w.h.p there is a constant fraction of the variables set wrongly before the unassignment phase begins.*

**Proof.** W.l.o.g, let $x$ be a variable assigned true by the original satisfying assignment. We expect $x$ to appear $2d$ times positively and $1.5d$ times negatively. The number of postive/negative appearances of $x$ is a binomial random variable with the properties required by Claim 7.1. Thus, there is a constant probability that $x$ will appear less than $2d - 0.25d$ times positively and more than $1.5d + 0.25d$ times negatively (and that would make the majority vote wrong for $x$). Hence, we expect a constant fraction of the variables to be set wrongly by the majority vote. Concentration around the expectation can be proved as in Claim 5.1.

In our algorithm, the unassignment phase comes straight after the majority vote, thus we have the claim. In [21], there is a phase of improving the approximation before the unassignment phase begins. However, variables which appear in less than $5\epsilon d$ clauses are not affected by it. Again, due to Claim 7.1, there is a constant fraction of variables appearing in less than $5\epsilon d$ clauses, and out of which for a constant fraction the majority vote was wrong (assume w.l.o.g that $\varphi(x) = T$; given that $x$ appears in $k$ clauses, $k$ a constant, in every clause it appears positively w.p. $\frac{4}{7}$; thus Claim 7.1 can be used to prove the latter). ■

**Corollary 7.3.** *In both algorithms, w.h.p there is a constant fraction of the variables unassigned by the unassignment phase.*

**Proof.** Both Flaxman and we prove that w.h.p the unassignment phase does not leave wrongly assigned variables. By the claim, w.h.p before the unassignment phase begins, a constant fraction of the variables are set wrongly. Thus, w.h.p at least those variables are unassigned by the unassignment phase. ■

#### 7.2.2 Failing the unassignment phase

Let $C$ be the set of variables appearing in at least $3.5d - \epsilon d$ clauses. Let $C' \subseteq C$ be an arbitrary $\frac{1}{5}$ of $C$. The adversary adds clauses, using only variables of $C'$, until all variables of $C'$ appear in at least $3.5d + 2\epsilon d$ clauses ($\epsilon, d$ are the constants used in [21]). The adversary added at most $|C'|3\epsilon d/3 = |C|\epsilon d/5$ clauses (we divide by 3 since every clause contributes to 3 variables). The expected size of $I$ is $7dn/6$, we assume that this is the actual size of $I$. We also assume that the algorithm extracts $d$ by actually calculating $d = 6|I|/7n$. The new parameter that the algorithm will calculate is $d' = \frac{6}{7n}\left(\frac{7dn}{6} + \frac{|C|\epsilon d}{5}\right) \leq d(1+\epsilon/5)$. The algorithm will unassign any variable which appears in more than $3.5d(1+\epsilon/5) + \epsilon(1+\epsilon/5)d = 3.5d + 1.7\epsilon d + \frac{\epsilon^2 d}{5} < 3.5d + 2\epsilon d$ clauses.
By the Chernoff bound, w.h.p. $|C| > (1 - e^{-\Theta(d)})n \Rightarrow |C'| > n/6$ for large enough $d$. Since $C'$ is unassigned, we have a constant (independent of $d$) fraction of unassigned variables. This contradicts the properties of the unassignment phase.

### 7.2.3 Creating a large connected component

In this section we will describe an adversarial strategy to interconnect a subset of the unassigned variables to create a connected component of linear size (in $n$). As we mentioned previously, the adversary adding clauses might change the set of unassigned variables. We show a strategy that adds clauses s.t every variable that was previously unassigned, will remain unassigned.

Before presenting the strategy, we first need to refine Claim 7.2. In Claim 7.2 we prove that both in our algorithm and in Flaxman's w.h.p there is a constant fraction of the variables that are wrongly assigned. Refine the claim in the following way. In the part concerning our algorithm, consider variables whose majority vote is wrong with a gap of at least 2. In Flaxman's algorithm, refine the requirement to variables appearing less than $5\epsilon d - 2$ times. It is easy to see that Claim 7.2 still holds (just repeat the proof with the refined quantities).

We are now ready to describe the adversarial strategy. The strategy is the same for both algorithms. We will analyze it on Flaxman's algorithm. The analysis in our algorithm will be evident from the latter. The adversary runs the algorithm (Flaxman's or ours) up until the unassignment phase is over. From corollary 7.3, w.h.p a constant fraction of the variables are unassigned. In particular, variables whose majority vote is wrong with a gap greater than 2 and appear less than $5\epsilon d - 2$ times are unassigned. Moreover, w.h.p we have a linear number of such variables. Now add clauses to create a cycle on those variables. For every such variable, we add at most 2 clauses containing it. From the refined requirements it follows that after adding the clauses each such variable is still unassigned. Also note that w.h.p we have at most $2^{-\Theta(d)}n$ such variables.
To sum up, by adding at most $2^{-\Theta(d)}n$ clauses, the adversary admits an $\Omega(n)$-size cycle (in particular, a cycle is a connected component) induced by unassigned variables.

Since we added clauses, we need to change $d$ in all the calculations we had. However, we added at most $2^{-\Theta(d)}n$ clauses, thus changing $d$ by at most $2^{-\Theta(d)}$. We can neglect this addition.

# 8 Properties of a Semi-Random Instance

Throughout the discussion in this section, whenever we refer to an instance $I$, we mean a semi-random instance $I$ whose random part belongs to $\mathbb{P}_{d,n}$.

We will start by showing that the properties we listed in section 5 also hold w.h.p. for the random part of the semi-random instance. The proof of the latter would be almost immediate. In the proofs we will use the parameter $d$. We refer to the $d$ used to define $\mathbb{P}_{d,n}$.
Next we show that the random walk converges in polynomial time (in the subformula induced by unassigned variables) even when adversary clauses are added.
We say that a semi-random instance is *typical* if its random part is typical. In the next section we analyze the behavior of the algorithm on a typical semi-random instance.

### 8.1 The Majority Vote

**Claim 8.1.** *Let $X$ be a random variable counting the number of variables in $I$ for which the majority vote disagrees with the original assignment. Then,*

$$Pr[X > 2^{-d/20}n] \leq e^{-\Theta(n)}$$

Same proof as in Claim 5.1 is valid here, since the adversary adds clauses that only push the majority in the correct direction. Namely, if for a variable the majority, restricted to $I_R$, agreed with the original assignment, then it would also agree w.r.t $I$.

### 8.2 The Core Variables

The definition of a core does not change in the semi-random model, since we defined a core w.r.t to an instance $I$, and an arbitrary assignment $\psi$, independent of the model in which $I$ was generated.

**Claim 8.2.** *Let $I$ be a semi-random instance. Let $\psi$ be an arbitrary assignment whose hamming distance from $\varphi$, the original satisfying assignment of $I$, is less than $3 \cdot 2^{-\alpha d}n$. Let $C$ be a core group of $I$ w.r.t to $\psi$. Then, w.h.p for every $x \in C$, $\psi(x) = \varphi(x)$.*

Before proving the claim we need the following lemma. As one will see later on, this lemma is an important key to understanding why the algorithm also works in the semi-random model.

**Lemma 8.1.** *Let $I$ be a semi-random instance with $\varphi$ its planted assignment. Let $\tau$ be an arbitrary assignment. Let $x$ be a variable whose assignment under $\tau$ disagrees with $\varphi$. Then, $x$ supports only random clauses of $I$ w.r.t $\tau$.*

**Proof.** Suppose w.l.o.g that $\tau(x) = T$. Thus, in every clause that $x$ supports w.r.t $\tau$, $x$ appears as a positive literal . Since $\varphi(x) = F$, in all the clauses of the adversary, $x$ appears as a negative literal (from the restriction we put on the adversary). Since in all adversary clauses the literal of $x$ evaluates to $F$ under $\tau$, all the clauses that $x$ supports w.r.t $\tau$ must be random ones. ■

*Note* that if we allow the adversary to add clauses in which at least 1 variable is a false literal under $\varphi$, this lemma is not necessarily true.

Due to Lemma 8.1, same proof as in Claim 5.3 is valid for Claim 8.2 as well.

Let $H(I_R)$, be the set variables constructed as in section 5.3, when regarding only clauses of $I_R$. Note that the core variable are defined only w.r.t the random part.

**Claim 8.3.** *W.h.p $H(I_R)$ is a core of $I$ w.r.t to the majority vote.*

**Proof.** $H$ is defined only on $I_R$. By observing that by adding clauses, the adversary can't push the majority in the wrong direction, it follows from Claim 5.4, that w.h.p $H$ is a core of $I_R$. It is easy to see that since $I_R \subseteq I$, $I$ is defined on the same set of variables as $I_R$, and the adversary can't reduce the support of a variable, $H$ is also a core of $I$. ∎

**Claim 8.4.** *All variables of $H(I_R)$ are set correctly*

Same proof as in Claim 5.6 is valid here, due to the following claims for the semi-random model: 8.1 (majority vote), 8.3 ($H$ is a core), and 8.2 (the core is set correctly).

### 8.3 The Connected Components

**Definition 8.1.** *Let $G_R(U, E)$ be the graph defined as follows. $U$ is the set of variables not in the core ($U = V \setminus H(I_R)$). $(x_i, x_j) \in E$ if there exists some clause in $I_R$ containing both $x_i, x_j$.*

**Claim 8.5.** *Given that $|U| < 3 \cdot 2^{-\alpha d} n$, the largest connected component in $G_R(U, E)$ is of size $\leq \log n$ w.h.p.*

**Proof.** In subsection 5.3 we proved that w.h.p the connected components in the graph induced by $V(I_R) \setminus H(I_R)$ are of size at most $\log n$. $H(I_R)$ is the same group of variables that we would have gotten if no adversary acted (since we build $H$ only w.r.t $I_R$). Hence, the discussion in subsection 5.3 is also valid here (w.r.t $I_R$). ∎

### 8.4 The Random Walk

**Claim 8.6.** *Consider a satisfiable $3CNF$ formula, $f$, over $n$ variables, whose largest connected component is of size at most $\log n$. Let $\varphi$ be a satisfying assignment. An arbitrary set of 3-satisfiable clauses w.r.t $\varphi$ are then added to receive $f'$. Then, using at most $(3t+1)n$ steps, the random walk procedure finds a satisfying assignment, for $f'$, w.p. $> 1 - \frac{n}{\log n} e^{-t/n}$.*

**Proof.** The connected components in $f$ are of size at most $\log n$. However, these components could be interconnected by adversary clauses. Hence we cannot assume that the connected components in $f'$ are of size at most $\log n$.

First consider a random walk taking place only on adversary clauses (namely in $f' \setminus f$). Again we start at distance at most $n$ from $\varphi$ and aim to reach distance 0 (or reach another satisfying assignment before reaching $\varphi$). Each time we look at an unsatisfied clause we improve with probability 1 (since all three variables are true under $\varphi$). In addition, we observe that when we flip a variable we cannot have another clause become unsatisfied. If we had such clause, it would mean that we have a variable appearing both as positive and negative (which contradicts the restriction we had on the adversary).

For conclusion, in every step we reduce the hamming distance between the current assignment and $\varphi$, and don't ruin any other clauses. Hence, in at most $n$ steps we reach $\varphi$.

Now look at the random walk in a connected component $\Gamma$ of $f$. The random walk in this component is affected only from decisions inside the component or decision regarding adversary clauses.

Whenever we examine an adversary clause, it might be the case that after we flip the variable we ruined some clauses in $\Gamma$. However, those clauses would have been ruined anyway (unless maybe another satisfying assignment is reached before $\varphi$) during the random walk in $\Gamma$, since their current assignment is wrong (at least for the variable that was flipped). Hence, we can view the random walk in every connected component as a random walk with "guiding tips toward $\varphi$" that are given to us every now and then.
For conclusion, the discussion of subsection 5.4 is also valid here (w.r.t $f$). Namely, w.h.p. the random walk in $f$ will converge to $\varphi$ within at most $3tn$ flips (unless we reach another satisfying assignment before reaching $\varphi$). After at most another $n$ flips we will also satisfy all remaining unsatisfied adversary clauses.
All in all, after $3tn + n$ flips, the random walk will reach a satisfying assignment for $f'$. ■

### 8.5 Probability for a Typical Semi-Random Instance

The bound on the probability remains the same as in the random case, since all properties have the same analysis.

## 9 Algorithm Analysis on a Typical Semi-Random Instance

Recall that the algorithm is parameterized with a parameter $d'$ (used in the unassignment phase). When running the algorithm on a semi-random instance, we give it $d'$ which refers to the $d$ used to define $\mathbb{P}_{d,n}$. If $d \geq c_0 \log n$, then w.h.p the majority vote set correctly all variables (recall that the adversary adds clauses that only push the majority in the correct direction). Otherwise, when running for $d' = 1...c_0 \log n$, in one of the iterations we will run with the correct $d'$. The following analysis is for the iteration where $d' = d$.

### 9.1 The Unassignment Phase

**Claim 9.1.** *After the unassignment phase ends, all variables in $H(I_R)$ remain assigned*

Same proof as in Claim 6.2 is valid here since $H(I_R)$ is a core of $I$ w.r.t the majority vote (Claim 8.3).

**Claim 9.2.** *After the unassignment phase ends, all assigned variables are assigned correctly.*

Due to Lemma 8.1, same proof as in Claim 6.3 is valid for this claim as well.

## 9.2 The Connected Components

**Definition 9.1.** *Let $G'_R(U', E')$ be the graph defined as follows. $U'$ is the set of* unassigned *variables in $I_R$. $(x_i, x_j) \in E'$ if there exists some clause in $I_R$ containing both $x_i, x_j$.*

**Claim 9.3.** *The largest connected component in $G'_R(U', E')$ is of size $\leq \log n$ w.h.p.*

Using the same reasoning as in Claim 6.4 (which is applicable due to Claims 8.5, 9.1) we get that the largest connected component in $G'_R$ is of size at most $\log n$.

## 9.3 Unit Clause Propagation

The analysis in the random setting (subsection 6.3) is also valid in the semi-random setting, due to Claim 9.2.

## 9.4 The Random Walk Phase

**Claim 9.4.** *Consider the subformula induced by unassigned variables. Using at most $(3t + 1)n$ steps, the random walk procedure finds a satisfying assignment for the subformula.*

**Proof.** First observe that the remaining subformula is satisfiable (since all assigned variables were set correctly, and $I$ is satisfiable). Our claim follows immediately by combining Claims 8.6 and 9.3. ∎

In subsection 5.4 we mentioned two variants of the random walk - with or without restart. The random walk with restart (for which we can prove better bounds on the success probability given the same number of steps) depends on the size of the connected components. When considering a semi-random setting, it is not true that w.h.p the connected components are of size at most $\log n$ (since an adversary can connect components or add new ones at a size of his choice). Hence, the random walk with restart procedure is less useful in the semi-random setting.

## 9.5 Success Probability

We analyze the success probability of the algorithm on a semi random instance, $I$, whose random part is sampled randomly from $\mathbb{P}_{d,n}$.

Let $R$ be the event that the RWalkSat procedure, performed on the subformula induced by unassigned variables, converges to a satisfying assignment in polynomial time.

$$Pr[\text{The algorithm finds a satisfying assignment for } I] \underbrace{\geq}_{\text{section 9}}$$

$$Pr[I \text{ is typical} \wedge R \text{ happens}] \underbrace{\geq}_{\text{subsection 8.5 and Claim 9.4}} 1 - n^{-\Theta(d)} - \frac{n}{\log n}e^{-t/n}.$$

Note that this is exactly the same bound we had for the random case.

# 10 A Deterministic Version of the Algorithm

We present a deterministic version of the algorithm. The deterministic algorithm may seem quite different at first glance, but its correctness stems from the same principles as the randomized version.

## 10.1 The Algorithm

**Definition 10.1.** *Let $\tau$ be an arbitrary assignment over the variables $\{x_1, x_2, ..., x_n\}$. Let $S$ be a subset of the variables. We denote by $\tau^{(S)}$, the assignment resulting from flipping the assignments of the variables of $S$ in $\tau$.*

**Definition 10.2.** *Let $I$ be a $3CNF$ over the variables $\{x_1, x_2, ..., x_n\}$. Let $T \subseteq I$ be an arbitrary set of clauses. We say that $P \subseteq \{x_1, x_2, ..., x_n\}$ is a* correcting path *for $T$ w.r.t an assignment $\tau$, if $\tau^{(P)}$ satisfies $T$.*

Note that we call $P$ a path although it is (an unordered) set of variables. The motivation for this name will become clear shortly.

Let $\varphi$ be the original satisfying assignment of the instance $I$.
Let $\psi = \psi_0$ be the assignment according to the majority vote over $I$.
Let $\psi_i$ be the assignment in the $i$'th iteration of the algorithm.
Let $S_i \subseteq I$ be the set of clauses satisfied by $\psi_i$. Let $U_i = I \setminus S_i$.

1. *Assign the variables according to $\psi$.*
2. $i \leftarrow 0$.
3. *While ($U_i \neq \emptyset$)*
   - *Pick a clause $c \in U_i$.*
   - *Set $T \leftarrow S_i \cup \{c\}$.*
   - *Find a correcting path, $P$, of length at most $\log n$ for $T$. If none found, fail the algorithm.*

- *Set* $\psi_{i+1} \leftarrow \psi_i^{(P)}$.
- *Set* $S_{i+1}$ *to be all clauses satisfied by* $\psi_{i+1}$, $U_{i+1} \leftarrow I \setminus S_{i+1}$.
- $i \leftarrow i+1$.

Consider an iteration of the algorithm for a not satisfied clause $c \in U_i$. We will show a procedure that finds a correcting path of length at most $\log n$ for $S_i \cup \{c\}$ in at most $1.5n$ steps (if one exists). The procedure is dynamically growing a search tree of depth at most $\log n$. Every node in the tree is a clause in $S_i \cup \{c\}$ that we need to resatisfy. Except for the first clause ($c$), we have 2 options to correct each unsatisfied clause (we don't flip a variable more than once, hence in every unsatisfied clause we have 2 more variables to try). We have a tree of depth $\log n$ in which every node is of degree at most 2 (except for the root). It follows that the tree has at most $3 \cdot 2^{\log n - 1} = 1.5n$ nodes. At the worst case we will scan the whole tree, thus using at most $1.5n$ steps.
Since the tree covers all correcting paths of length at most $\log n$, if there is a correcting path of that size we are sure to find it.
The name *path* is now evident from the procedure we described for finding the correcting subset of variables - the variables form a path in the tree.

Note that the naive way to find a correcting path, of going over all possible subsets of variables of size $\log n$, could take time $\binom{n}{\log n} \approx n^{\log n - \log\log n}$. This of course turns the algorithm into a non-polynomial one.

## 10.2 Algorithm Analysis

**Claim 10.1.** *Let $I$ be a typical instance. Let $c \in U_i$ be a clause not satisfied by $\psi_i$. Then, there is* no *correcting path of length at most* $\log n$ *for* $S_i \cup \{c\}$ *containing a variable from the core group (H).*

**Proof.** Suppose by contradiction that there exists such a path. Let $i$ be the first iteration in which such a path exists. Since up until now we haven't flipped any variable from the core (since no correcting path containing core variables existed) every variable in the core supports at least $\frac{d}{2} - 2\epsilon d$ clauses containing only other core variables. Let $W$ be the set of core variables belonging to the correcting path. Whenever flipping $x \in W$, every clause that it supports w.r.t $\psi_i$ either becomes not satisfied, and in that case we need to flip another variable $x' \in W$ to resatisfy the clause, or if the clause remains satisfied, it means that we already flipped some variable $x'' \in W$ in an earlier stage of the path. In either case, every clause that $x$ supports w.r.t $\psi_i$ contains 2 variables from $W$.
Since $i$ is the first iteration that we flip variables from the core, every variable in $W$ supports at least $\frac{d}{2} - 2\epsilon d$ clauses w.r.t $\psi_i$. Since the support is unique, every such clause is counted once.
To sum up, $|W| \le \log n$, $\frac{d}{2} - 2\epsilon d > 2$, and we have $|W|(\frac{d}{2} - 2\epsilon d)$ clauses containing two variables from $W$. Since $I$ is typical this cannot happen (Lemma 5.2). ■

**Claim 10.2.** *Let $I$ be satisfiable. Let $c \in I$ be a clause not satisfied by $\psi_i$. Let $x$ be a variable in $c$ whose assignment disagrees with $\varphi$. Then, by flipping the assignment of $x$ no additional clause in $I_A$ (the adversarial clauses) becomes unsatisfied.*

**Proof.** W.l.o.g assume that $x$ appears positive in $c$. Namely, $\psi_i(x) = F$ and $\varphi(x) = T$. In all adversarial clauses, $x$ appears as a positive literal (from the restriction we had on the adversary). Hence, by flipping $x$'s assignment from $F$ to $T$ we cannot have adversarial clauses become unsatisfied. ■

**Corollary 10.3.** *There exist a correcting path for $S_i \cup \{c\}$ induced only by random clauses of $I$ (except maybe for the first clause). Moreover, if $I$ is typical, the path does not contain core variables.*

**Proof.** Consider the path starting by flipping $x$. By the claim, we know that all clauses that become unsatisfied in $S_i$ are random ones. Since $x$ in now set correctly, every unsatisfied clause must contain another variable whose assignment disagrees with $\varphi$. Continue the path by flipping the wrongly assigned variables. Since $S_i$ did not have unsatisfied clauses to begin with, and $c$ will not become unsatisfied again (since we don't flip a variable more than once), it follows by induction that all the clauses that induce the path are random. Since $I$ is satisfiable, and we flip wrongly assigned variables, this procedure is sure to find a correcting path.
If $I$ is typical, then the core variables are set correctly by the majority vote (Claim 8.4). Our path contains only wrongly assigned variables, thus it does not contain any core variables. ■

**Claim 10.4.** *For a typical satisfiable semi-random instance, the algorithm finds a satisfying assignment.*

**Proof.** We need to show that in every iteration there exists a correcting path of size at most $\log n$. Consider the $i$'th iteration. Let $c$ be the not satisfied clause that was added to $S_i$. $c$ is not satisfied by $\psi_i$ but is satisfied by $\varphi$. Hence, $c$ contains a variable $r$ which is set incorrectly by $\psi_i$. Consider the search tree whose root is $r$. The algorithm does not flip core variables at any stage (Claim 10.1), thus they remain correctly assigned throughout the algorithm (Claim 8.4). Furthermore, $I$ is typical and satisfiable. By corollary 10.3, we know that there exists a correcting path starting at $r$ consisting only of random clauses, and not containing any core variable. Combining these facts with Claim 8.5 (regarding the size of the connected components in the graph induced by non-core variables), we get that the path is of length at most $\log n$ (at most the size of the connected component of $r$ in $I_R$). The procedure for finding a correcting path scans all paths of length $\log n$, thus it is sure to find a correcting path (at least the one promised by corollary 10.3). ■

# 11 General Notes

## 11.1 The Deterministic Algorithm

Consider the following variant of the algorithm we presented in section 10. Instead of limiting the correcting path to be of length at most $\log n$, change the rule to finding the shortest correcting path. By doing so, we get a simple algorithm that always succeeds in finding a satisfying assignment for every distribution of $3CNF$s (whenever the input is satisfiable). Moreover, for our semi-random distribution, we proved that this algorithm works w.h.p in polynomial time (since w.h.p the shortest correcting path is of length at most $\log n$).

## 11.2 Why does the random walk work ?

We go back to discussing the probabilistic version of the algorithm (section 3). It is shown in [4], that the random walk algorithm (the last phase of our algorithm) takes w.h.p. exponential time when running on a random $3CNF$ from the "planted sat" distribution (starting from a random assignment). They start by showing that it does poorly on a $3CNF$ composed of all clauses satisfied by a certain assignment (called a *full* $3CNF$). Then they show that the behavior of the random walk on a random part of the full $3CNF$, is w.h.p. the same as on the full $3CNF$.
We do a random walk over $O(n)$ variables and w.h.p it converges in polynomial time. Where does the analysis of [4] go wrong in our case ?
Two crucial things are different from the case [4] analyzed. The first is that we don't start from a random assignment. Rather, we start with an assignment which is correct for a large fraction of the variables. The second is that we can identify the variables which are set correctly (those surviving the Unassignment phase), and don't flip them. As we saw, those two ingredients cause the remaining formula to break into small connected components and let the random walk converge in polynomial time.

## 11.3 The semi-random model

### 11.3.1 Why did we add 3-satisfiable clauses ?

The main motivation for 3-satisfiable clauses is the fact that we consider them "easy" clauses (as they are easy for many natural 3SAT heuristics). As such, they are a good benchmark to see if an algorithm is natural, robust, not over-fitted to the model in which we generate its instances, etc.

#### 11.3.2 Why didn't we allow the adversary to add also other kinds of clauses ?

We would have liked to do that, but at present we just could not find an algorithm that handles such an adversary.

#### 11.3.3 How can such an adversary fool our particular algorithm ?

If we allowed the adversary to add other kinds of clauses than we could have had problems with the successes rate of the majority vote, the correctness of the core variables and the convergence in polynomial time of the random walk.

## 12 Test Results

We go back to discussing the randomized version of the algorithm (the one presented in section 3). We coded the algorithm in C++. We also coded an algorithm for generating random instances in a variant of the planted distribution (details ahead). Recall that the planted distribution is parameterized with two main parameters. The number of variables $n$, and the probability for including each satisfiable clause, $p = d/n^2$. While checking up to $n = 10^6$, the lowest $d$ that the algorithm worked for was 14.57 (i.e. a clause to variable ratio of 17). The analysis predicts a somewhat higher ratio. In this section we will describe the tests that lead us to this empirical result.

### 12.1 Redefining the distribution

Let $\mathbb{P}_{d,n}$ be the distribution defined in section 2. Namely, picking at random a truth assignment for the $n$ variables and including each clause satisfied by it w.p. $d/n^2$.

We now define a new distribution $\mathbb{S}_{c,n}$. The new distribution contains formulas generated by picking at random a truth assignment and then picking at random (without repetitions) $cn$ clauses satisfied by it.

Note that the natural relation between the two distributions is given by $c = 7d/6$, since the expected size of an instance sample from $\mathbb{P}_{d,n}$ is $(7d/6)n$. Unless stated otherwise, throughout the discussion in this section we refer to the case when $c = 7d/6$.

The motivation for using $\mathbb{S}_{c,n}$ instead of $\mathbb{P}_{d,n}$ in the tests is as follows. If we took the naive approach for generating an instance in $\mathbb{P}_{d,n}$, it would have taken time $O(n^3)$. This makes generating instances with more than a few hundreds of variables practically impossible.

We would like to have a distribution where we can generate an instance in time $O(m)$ (where $m$ is the number of clauses, which is linear in $n$ in our case). $\mathbb{S}_{c,n}$ enables us to do that quite easily, as we will see in the next subsection.

### 12.1.1 Generating an Instance

We describe the procedure we used for generating an instance in $\mathbb{S}_{c,n}$.
*The inputs are*: a truth assignment $\varphi$, the number of variables - $n$, and the number of clauses - $m = cn$.
*The output is* a formula which has at least one satisfying assignment ($\varphi$).
*The procedure* for generating a random satisfiable clause is simple. We pick 3 numbers in $[1, n]$, which will index the three variables (if we happen to choose the same number twice, we repeat the procedure). Every 3 variables have 7 ways of complementing them so that $\varphi$ would satisfy that clause. We list all 7 possibilities and pick one at random. We sign the variables according to it.
The procedure for generating a formula is simply repeating the procedure for generating a clause $m$ times.

$\mathbb{S}_{c,n}$ does not contain formulas with the same clause appearing more than once. Our procedure may admit such instances. Taking care of this requirement slows down the sampling algorithm (and the naive way would raise the running time to $O(m^2)$). One can easily verify that the probability for a formula with a clause appearing twice is low, $O(\frac{c^2}{n})$, using standard second moment calculations. Thus, we can allow ourselves to neglect this issue and assume that the procedure indeed samples a random instance from $\mathbb{S}_{c,n}$.

### 12.1.2 A word on efficiency

Suppose $m = cn$ for some constant $c$. For a random clause (as generated above), with probability $\binom{3}{2}\frac{1}{n} = \frac{3}{n}$ we will have a variable which appears twice. Hence, we expect $3c$ clauses to be defective (tightness could be proved using the Chernoff bound since clauses are generated independently). For those clauses we run the procedure again. With probability at least $1 - \frac{9c}{n}$, all clauses would be corrected (union bound). Hence, we expect the procedure to run in time $O(m)$ w.h.p.

## 12.2 Comparing the two distributions

$\mathbb{P}_{d,n}$ is not the same distribution as $\mathbb{F}_{d,n}$. In particular, in $\mathbb{S}_{c,n}$ all formulas have fixed length and in $\mathbb{P}_{d,n}$ no. However, the two distributions are close enough so that all properties that may interest us, are common to both. In particular, we expect the algorithm to work on the new distribution as well. For further discussion in similar contexts see [9].

## 12.3 Cross-checking the majority vote

We would like to verify that the majority vote success rate that we see is indeed what we expect (since we used the random method of the computer). We wrote a program to calculate (using brute-force) the probability of the majority vote being correct for a single variable. From the linearity of

expectation, this is exactly the expected fraction of variable for which the majority vote is correct. To simplify calculations, we will analyze the probability on an instance sampled from $\mathbb{P}_{d,n}$ (as mentioned before, it doesn't really matter on which distribution we analyze this property).
For a variable $x$ assigned $T$ by the original assignment, let $X_T$ be a r.v. counting the number of true appearances, and $X_F$ the number of false.

$$X_T \sim B(p, 2n^2),\ X_F \sim B(p, 1.5n^2)$$

the probability of the majority vote being right is exactly

$$Pr[X_T > X_F] + \frac{1}{2} \cdot Pr[X_T = X_F] = \sum_{i=0}^{2n^2} Pr[X_T = i] \cdot \left( \frac{1}{2} Pr[X_F = i] + \sum_{k=0}^{i-1} Pr[X_F = k] \right)$$

The half factor comes from the fact that when we have a tie, no matter how we break it we have probability $\frac{1}{2}$ of being wrong (here we cannot assume that $x$ was assigned true).
Calculating directly this sum is very resource-consuming and in fact impractical when doing it naively. We added a few optimizations that enabled us to efficiently calculate the value with an o(1) error.
The main optimization is summing up to $O(d \log n)$ instead of $2n^2$. Using Chernoff bounds, one can easily prove that all but a $o(1)$ of the probability mass, is concentrated below $O(d \log n)$.

## 12.4 Test results

A few notes regarding the structure and content of the table.

- The algorithm is parameterized with the parameter $d$ and a slackness parameter $\epsilon$. Every variable is expected to support $\frac{d}{2}$ clauses. The relaxed support that the algorithm looks for is $(\frac{d}{2} - 2\epsilon d)$. In the result table we denote this value by *actual support* ($\epsilon$ can be derived from that value).

- The last column indicates what was the success rate in all the tests.

- In the column showing the expected majority vote success rate the format is $a \pm b$. $a$ is the normalized (i.e. divided by $n$) expected success rate calculated by the brute-force calculation program we wrote. $b$ is one normalized standard deviation.

- Whenever there are two values in the *unassigned fraction* column, the first is the fraction of variables unassigned by the unassignment phase of the algorithm. After the unassignment phase ends, it might be the case that wrong variables remain assigned. In particular, we might have unsatisfied clauses whose all three variables remain assigned. In this case, we either fail the algorithm, or have a further unassignment phase (unassign all variables in such clauses) and go on with the algorithm. The second value is the further unassigning we did to avoid contradictions.

- The unassignment, simplification and #flips are averages over the successful tests only.
- Let $d = \frac{6c}{7}$, The expected support for a variable is $\frac{d}{2} = \frac{3c}{7}$. The expected size of formula is $cn$. We sampled (using the procedure described above) formulas of size exactly $cn$, where $c$ is called the ratio.
- Reassigned fraction column refer to variables reassigned (after the unassignment phase unassigned them) by the unit clause propagation rule.
- All the unassigned and reassigned fractions are taken from the total number of variables, $n$. Using the Chernoff bound, one gets that a variable will not appear w.p. at most $4 \cdot 10^{-11}$ for $c = 17$. In all our tests $n \leq 10^6$. Hence, w.p. at most $4 \cdot 10^{-5}$ we will find a variable which doesn't appear. Thus, one can think of the instances as containing all variables.

  Note that for a larger $n$, we expect a linear fraction of the variables *not* to appear. This property (of all variables appearing) is more suitable for the case that the clause to variable ratio is $c \log n$ for some small constant $c$ ($c$ cannot be too large, since another property characterizing this ratio does'nt hold - the majority vote is still wrong for linear fraction of the variables).

| #vars | maj. succ. rate | expected maj. succ. rate | unassigned fraction | reassigned fraction | RWalk #flips | succ. rate |
|---|---|---|---|---|---|---|
| c=17, expected support=7.29, actual support=2, averaging on 10 tests | | | | | | |
| 1000 | 0.85 | $0.846 \pm 0.011$ | 0.1899 | 0.1893 | 0 | 1 |
| 2000 | 0.84 | $0.846 \pm 0.008$ | 0.2189 | 0.2183 | 0 | 1 |
| 5000 | 0.849 | $0.846 \pm 0.005$ | 0.203 | 0.202 | 0 | 1 |
| $1 \cdot 10^4$ | 0.846 | $0.846 \pm 0.003$ | 0.209 | 0.208 | 0 | 1 |
| $2 \cdot 10^4$ | 0.846 | $0.846 \pm 0.002$ | 0.209 | 0.208 | 0 | 1 |
| $5 \cdot 10^4$ | 0.847 | $0.846 \pm 0.0015$ | 0.212 | 0.211 | 0 | 1 |
| $1 \cdot 10^5$ | 0.847 | $0.846 \pm 0.001$ | 0.208 | 0.207 | 0 | 1 |
| $2 \cdot 10^5$ | 0.847 | $0.846 \pm 0.0008$ | 0.210 | 0.209 | 0 | 1 |
| $5 \cdot 10^5$ | 0.8468 | $0.846 \pm 8.29 \cdot 10^{-4}$ | 0.208 | 0.2077 | 0 | 1 |
| c=17, expected support=7.29, actual support=2, averaging on 1 test | | | | | | |
| $1 \cdot 10^6$ | 0.8477 | $0.846 \pm 8.29 \cdot 10^{-4}$ | 0.206 | 0.205 | 5 | 1 |
| c=16, expected support=6.85, actual support=2, averaging on 10 tests | | | | | | |
| 1000 | 0.83 | $0.84 \pm 0.011$ | 0.259 | 0.258 | 0 | 0.8 |
| 2000 | 0.84 | $0.84 \pm 0.008$ | 0.256 | 0.255 | 0 | 0.8 |
| 5000 | 0.839 | $0.84 \pm 0.005$ | 0.265 | 0.264 | 0 | 0.7 |
| 10000 | 0.838 | $0.84 \pm 0.004$ | 0.287 | 0.286 | 0 | 0.5 |
| c=16, expected support=6.85, actual support=1, averaging on 10 tests | | | | | | |
| 1000 | 0.834 | $0.84 \pm 0.011$ | 0.04,0.4 | failed | 0 | 0 |
| 5000 | 0.842 | $0.84 \pm 0.005$ | 0.04,0.4 | failed | 0 | 0 |

For $c < 16$, the algorithm fails, either by unassigning all variables or by leaving wrong variables after the unassignment phase ends. In some machines there might be a shift in the value of $c$ (e.g, the algorithm will work for $c \geq 18$ and fail for c=17). This is due to differences in the random mechanisms of different machines.

Note that in all entries, the reassigned fraction is less than the unassigned fraction, and the formula was satisfied without random walk. This means that all remaining unassigned variable, appear in clauses that are already satisfied (and hence were simplified). Note that this also means that the formula had more than one satisfying assignment. In [7], it is proved that when $c = \Delta \log n$ for large enough constant $\Delta$, the instance would have w.h.p exactly one satisfying assignment. One can easily use that proof to show that when $c = O(1)$, the expected number of satisfying assignments of hamming distance $k$ from $\varphi$ is roughly $\binom{n}{k} e^{-ck/2}$, for $k \ll n$.

## 12.5 Running on DIMACS instances

### 12.5.1 The probability space

A literal $l$ is *entailed* by a satisfiable SAT instance $f$ iff $f \wedge \bar{l}$ is unsatisfiable. The *backbone* of a satisfiable SAT instance is the set of entailed literals and so the backbone size is the number of entailed literals. Each instance has 4 parameters which define the distribution from which it was sampled. $k$: the number of literals per clause (3 in our case), $n$: the number of variables, $m$: the number of clauses, $b$: the size of the backbone.
Each instance is generated as follows:

- Generate $m$ random clauses by for each clause, select at random 3 distinct variables from the set of $n$. Negate each with probability $\frac{1}{2}$. These form the literals of the clause.
- If the instance is unsatisfiable, restart the generation process.
- Determine the backbone size. If it is not equal to $b$, restart the generation process.

### 12.5.2 Test results

All instances had 100 variables and between 400-500 clauses. Our algorithm failed on all instances. The reason for the failure was that all variables were unassigned (even when unassigning only variables whose support is 0). However, we did not expect our algorithm to work on such sparse instances since:

- The majority vote success rate is very small.
- In such sparse formulas every variable is expected to support 1-2 clauses. Hence, we don't have enough slackness for the algorithm to work. Either all variables are unassigned or none. In both cases the algorithm fails.

We ran the algorithm on instances from $\mathbb{P}_{d,n}$, and $\mathbb{S}_{c,n}$ with the same clause to variable ratio ($c = 4, 5$ and $d = 6c/7$). In this case, we were also able to measure the success rate of the majority vote (unlike the DIMACS case, we had the original satisfying assignment). The success rate was about 65%. The algorithm failed also on planted instances since it unassigned all variables.

## 12.6 Testing the Deterministic Version

The main flaw of the deterministic version is that its running time (in the worst case, where we need to scan the whole search tree in every iteration) is exponential in the size of the connected components. Recall that in every iteration we dynamically grow a search tree of depth at most

$\log n$ in which every node has degree 2. When running for low clause to variable ratios, we are not guaranteed w.h.p to have connected components of size $\log n$. And indeed, the algorithm fails when allowing it run up to a depth of $\log n$ for low ratios (e.g., for instances in $\mathbb{S}_{c,n}$, for $c < 30$). However, allowing it to run up to a depth of $2\log n$ for example, already increases the running time of each iteration to $n^2$. This makes testing practically impossible for large $n$ (large enough so that $c$ could be thought of as a constant w.r.t $n$).

# 13 Future Work

In this section we describe several interesting problems that may be a good starting point for future work in the area. In the scope of this work we did not have time to address those issues.

## 13.1 Unit Clause Propagation Success

As one can see in the test result tables, when running for $c = 17$, and $n \leq 5 \cdot 10^5$, we did not need the random walk phase. The unit-clause-propagation found the satisfying assignment for the subformula induced by unassigned variables. However, for $n = 10^6$ we already needed a few steps of the random walk. An interesting property to prove is that the unit clause propagation (maybe combined with another heuristic such as the pure-literal heuristic [12]) finds w.h.p a satisfying assignment for the subformula induced by unassigned variables.

## 13.2 Random Walk Efficiency

When running the randomized version of the algorithm without the unit-clause-propagation (namely, starting the random walk immediately after the unassignment phase ends), we noticed that the random walk works in time that seems to be linear in $n'$, where $n'$ is the number of unassigned variables (typically $n' = 2^{-\Theta(d)}n$). E.g., for $c = 18$, we observed that the random walk converged in time roughly $2n'$, $n' = 0.18n$.

Our analysis of the random walk demands at least $n^2$ steps. An empirical intuition for this gap is the observation that most clauses in the formula induced by unassigned variables are of length 1 or 2. Suppose that the subformula was entirely $2SAT$. It is shown in [30] that the random walk converges w.h.p in at most $O(s^2)$ steps (where $s$ is the number of nodes in the random walk line). Thus, using at most $O(\log^2 n)$ steps, the random walk will converge in every connected component. Hence, we expect at most $\frac{n'}{\log n} \log^2 n = n' \log n$ steps in the random walk. However, this still doesn't explain the linear time.

It would be nice to rigourously prove that the random walk converges in at most a linear number of steps.

Also see the work in [4] regarding upper bounds on the random walk in other contexts.

### 13.3 Algorithm for smaller ratios

From our test results, it seems like the algorithm already works for clause to variable ratio of 17. Moreover, it is clear that for a ratio of 16 the algorithm fails. The lower the ratio is, the lower the majority vote success rate is. As a result, one should not expect logarithmic-size connected components when the majority is wrong for say 1/5 of the variables. Moreover, Claim 5.3, which is one of the basic motivations for the algorithm, might also become invalid (at least the analysis we gave is invalid for low ratios). It seems like a new algorithmic approach is needed to tackle such sparse formulas.

### 13.4 Another Semi-Random Model

Consider the following semi-random model. First a random formula is generated with a random planted assignment $\varphi$. Next, for every clause $c$ in the formula, the adversary can change the polarity of its variables. The only restriction is that the new polarity agrees with $\varphi$ (namely, the literal that was changed evaluates to $T$ under $\varphi$).

Now consider the following adversarial strategy. Take the subformula containing clauses with only core variables (w.r.t the majority vote). For every clause in the subformula, change polarity s.t it is satisfied by at least two variables. Due to that, all variables of the core would have support of at most $\epsilon d$ (since w.h.p every core variable supports at most $\epsilon d$ clauses with variables not in the core) and there is no gain to the majority vote success rate (since the core variables are set correctly to begin with). This strategy already fails both our randomized version and Flaxman's algorithm (since all core variables would be unassigned). It also impedes the analysis we gave for the deterministic version. In particular, it need not follow that the algorithm does not flip core variables (which is a basic key to our analysis).

It seems that for this semi-random model, another algorithmic approach is in due (or prove that there exists no "good" probabilistic polynomial time algorithm for this semi-random model).

## Acknowledgements

I would like to express my sincere gratitude to my thesis advisor professor Uriel Feige, for his guidance and support. Not only that his advice proved to be most helpful in many occasions, but also his door was always open for discussion and help.

I would like to thank my fellow students of the Weizmann CS department for making my time at Weizmann a very pleasant one, and for the long and interesting discussions on scientific and non-scientific issues. I would also like to thank Eli Ben-Sasson for his useful comments.

Last but not least, I am very grateful for the love and support of my wife Rolanda.

## References


[1] D. Achiloptas. Lower bounds for 3-SAT via differential equations. *Theoretical Computer Science*. 265 (1-2), pp. 159-185, 2001.

[2] D. Achiloptas, G.B. Sorkin. Optimal myopic algorithms for random $3SAT$. *IEEE Symposium on Foundations of Computer Science*. pp. 590-600, 2000.

[3] Martin Aigner, Gunter M. Ziegler. *Proofs from The Book*. Springer-Verlag Berlin Heidelberg, 1998.

[4] Mikhail Alekhnovich, Eli Ben-Sasson. Linear Upper Bounds for Random Walk on Small Density Random 3-CNFs. *FOCS 2003*, pp. 352-361.

[5] Noga Alon, Nabil Kahale. A spectral technique for coloring random 3-colorable graphs. *SIAM J. COMPUT.*, Vol. 26, No. 6,1733-1748. December 1997.

[6] N. Alon, M. Krivelevich, B. Sudakov. Finding a large hidden clique in a random graph. *Random Structures and Algorithms 13*, pp. 457-466. 1998.

[7] Eli Ben-Sasson, Yonatan Bilu and Danny Gutfreund. Finding a Randomly Planted Assigment in a Random $3CNF$. *manuscript*. May 2002.

[8] A. Blum, J. Spencer. Coloring random and semirandom $k$-colorable graphs. *J. Algorithms, 19*,pp. 204-234, 1995.

[9] Béla Bollobás. *Random Graphs*. Academic Press, 1985, chapter II.

[10] B. Bollobás, T.I Fenner, A.M Frieze. An Algorithm for finding Hamilton paths and cycles in random graphs. *Combinatorica 7(4)* pp.327-341, 1987.

[11] R. Boppana. Eigenvalues and graph bisection: An average-case analysis. *In Proceedings of the 28th Annual Symposium on Foundations of Computer Science*,pp. 280-285, IEEE Computer Society Press, Los Angeles, CA, 1987.

[12] A. Broder, A. Frieze, E. Upfal. On the satisfiability and maximum satisfiability of random 3-CNF formulas. *In Proceedings of the Foruth Annual ACM-SIAM Symposium in Discrete Algorithms*, pp. 322-330.

[13] B. Cha, K. Iwama. Performance Test of Local Search Algorithms Using New Types of Random CNF Formulas. *In Proc IJCAI-95 14th International Joint Conference in Artificial Intellignece*, Vol 1. pp. 304-310. Montreal, August 1995.

[14] Hui Chen, Alan Frieze. Coloring Bipartite Hypergraphs . *IPCO: 5th Integer Programming and Combinatorial Optimization Conference*, pp. 345-358, June 20, 2000.

[15] V. Chvàtal, E. Szemerèdi. Many Hard Examples for Resolution. *Journal of the Association for Computing Machinery* vol. 35, pp. 759-768, 1988.

[16] S.A. Cook. The Complexity of Theorem-Proving Procedures. *In Proc. 3rd IEEE FOCS*, pp. 151-158, 1971.

[17] J. Crawford, L. Auton. Experimental results on the crossover point in random 3SAT. *Artificial Intelligence 81* pp.31-57, 1996.

[18] Uriel Feige. Relations Between Average Case Complexity and Approximation Complexity. *In Proc. of STOC 2002*, Montreal.

[19] Uriel Feige, Joe Kilian. Heuristics for Semirandom Graph Problems. *Joural of Computer and System Sciences 63*, pp.639-671, 2001.

[20] Uriel Feige, R. Krauthgamer. Finding and certifying a large hidden clique in a semi-random graph. *Random Structures and Algorithms 16(2)*, pp.195-208, 2000.

[21] Abraham Flaxman. A spectral technique for random satisfiable $3CNF$ formulas. *SODA 2003*, pp.357-363, July 2002.

[22] E. Friedgut. Necessary and sufficient conditions for sharp thresholds of graph properties, and the $k$-SAT problem. *J. Amer. Math. Soc. 12*, pp.1017-1054, 1999.

[23] A. Frieze and C. McDiarmid. Algorithmic theory of random graphs. *Random Structures and Algorithms 10*, pp. 5-42, 1997.

[24] A. Kaporis, L. Kirousis, E. Lalas. The probabilistic analysis of a greedy satisfiability algorithm. *Proc. 5th Satisfiability Testing Workshop*, 362-376, 2002.

[25] A. Kaporis, L. Kirousis, Y. Stamatiou, M. Vamvakari, M. Zito. Cupon Collectors, q-Binomial Coefficients and the Unsatisfiability Threshold *Proc. 7th ICTCS*, 328-338, 2001.

[26] E. Keydar. Finding Hamiltonian Cycles in Semi-Random Graphs. *manuscript*. The Weizmann Institue of Science, Rehovot, Israel, November 2002.

[27] S. Kirpatrick, C.D Gelatt, M.P Vecchi. Optimization by simulated annealing. *Science 220*, pp. 671-680, 1983.

[28] M. Komolòs, E. Szemerèdi. Limit distribution for the existence of a Hamilton cycle in a random graph. *Discrete Mathematics*, 43, pp. 55-63, 1983.

[29] Elias Koutsoupias, Christos H. Papadimitriou. On the greedy algorithm for satisfiability. *Information Processing Letters*, 43, pp. 53-55, August 1992.

[30] Christos H. Papadimitriou. On selecting a satisfying truth assigment. *In Proceedings of the 32nd Annual IEEE FOCS'91* pp. 163-169.

[31] M. Santha and U.V. Vazirani. Generating quasi-random sequences from semi-random soruces. *JCSS*, 33:75-87, 1986.

[32] Uwe Schöning. A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems. In *Proc FOCS'99*, pp.410-414, 1999.

# 14 Appendix

## 14.1 Complete proof of Lemma 5.2

**Lemma 5.2.** *Let $c \geq 1$ be some constant. Then w.h.p (at least $1-O(n^{2-2c})$), there exists no subset of variables $U$, s.t. $|U| \leq 3 \cdot 2^{-\alpha d}n$ and there are at least $c|U|$ clauses containing two variables from $U$.*

**Proof.** For a fixed set $U$ with $|U| = k$, the number of clauses containing two variables from $U$ is

$$\binom{k}{2}(n-2)2^3 \leq 4k^2 n$$

Each of the clauses is included independently w.p. $\frac{d}{n^2}$. Thus, the probability that $ck$ of them are included is at most

$$\binom{4k^2 n}{ck}\left(\frac{d}{n^2}\right)^{ck} \leq \left(\frac{4k^2 ne}{ck} \cdot \frac{d}{n^2}\right)^{ck} \leq \left(\frac{12kd}{cn}\right)^{ck}$$

We complete the proof by a union bound. The probability that any set $U$ has this property is at most

$$(*) \sum_{k=2}^{3\cdot 2^{-\alpha d}n} \binom{n}{k}\left(\frac{12kd}{cn}\right)^{ck} \leq \sum_{k=2}^{3\cdot 2^{-\alpha d}n} \left(\frac{ne}{k}\right)^{k}\left(\left(\frac{12kd}{cn}\right)^{c}\right)^{k} =$$

$$= \sum_{k=2}^{3\cdot 2^{-\alpha d}n} \left(e\left(\frac{12d}{c}\right)^{c}\left(\frac{k}{n}\right)^{c-1}\right)^{k}$$

**Claim.** *Let $q \in (0,1)$ be a constant depending only on $d, c$. Let $a_k = \left(e\left(\frac{12d}{c}\right)^{c}\left(\frac{k}{n}\right)^{c-1}\right)^{k}$. Then $a_{k+1}/a_k < q$ for every $k \in [2, 3 \cdot 2^{-\alpha d}n]$.*

**Corollary.** *The probability for such a set $U$ is $O(n^{2-2c})$.*

**Proof.**(cor.) Since the sequence $\{a_k\}_{k=2}^{3\cdot 2^{-\alpha d}n}$ decreases faster than the geometrical series with parameter $q$, we can bound $(*)$ by $\frac{a_2}{1-q}$. Namely, by

$$\frac{1}{1-q}\left(e\left(\frac{12d}{c}\right)^{c}\left(\frac{2}{n}\right)^{c-1}\right)^{2} \leq n^{2-2c}\left(\frac{e12d}{c(1-q)}\right)^{2c} = O(n^{2-2c})$$

■

**Proof.**(claim) We will show that $a_{k+1}/a_k < q$ for $k \in [2, 3 \cdot 2^{-\alpha d} n]$.

$$\frac{a_{k+1}}{a_k} = \left(e\left(\frac{12d}{c}\right)^c \left(\frac{k+1}{n}\right)^{c-1}\right)^{k+1} \left(e^{-1}\left(\frac{c}{12d}\right)^c \left(\frac{n}{k}\right)^{c-1}\right)^k = e\left(\frac{12d}{c}\right)^c n^{1-c}\left(\frac{k+1}{k}\right)^{k(c-1)} (k+1)^{c-1}$$
$$= e\left(\frac{12d}{c}\right)^c \left(1+\frac{1}{k}\right)^{k(1-c)} \left(\frac{k+1}{n}\right)^{c-1} \le e^2 \left(\frac{12d}{c}\right)^c \left(\frac{k+1}{n}\right)^{c-1} \le \left(\frac{e^2 12d}{c}\right)^c \left(\frac{4 \cdot 2^{-\alpha d} n}{n}\right)^{c-1}$$
$$\le \left(\frac{360d}{2^{\alpha d} c}\right)^{c-1} \frac{90d}{c} \le \left(\frac{d^3}{2^{\alpha d}}\right)^{c-1} \le (\frac{1}{2})^{c-1} \text{ for large enough } d$$

Since $c > 1$ is a constant, $(\frac{1}{2})^{c-1}$ is some constant smaller than 1. ■

This finishes the proof of Lemma 5.2. ■

### 14.2 Complete proof of Lemma 5.4 and Claim 5.10

**Lemma 5.4.** *Let $F$ be a set of clauses and let $H(F \cup T')$ be that value of $H$ if $I = F \cup T$. Let $H'(F)$ be the value of $H'$ if $I = F$. Then $H'(F) \subseteq H(F \cup T')$.*

**Proof.** We prove the claim by induction. First we show that $H'_0(F) \subseteq H_0(F \cup T')$. If $x \in H'_0(F)$ then the majority vote for $x$ is right with advantage of at least 7, and $x$ supports at least $\frac{d}{2} - \epsilon d$ clauses. Since the degree of $x$ in $T'$ is at most 6, the majority vote could not be flipped by adding $T'$ to $F$. Also the support could only increase by adding $T'$, hence $x \in H_0(F \cup T')$ as well. We proceed by showing that during the iterative step we still have $H'_i(F) \subseteq H_i(F \cup T)$ for every $i$. Suppose that $H'_i(F) \subseteq H_i(F \cup T')$. We show that the claim holds for $i+1$. If $x$ supports at least $\frac{d}{2} - 2\epsilon d$ clauses with only variables of $H'_i(F)$ w.r.t. $F$, then since $H'_i(F) \subseteq H_i(F \cup T')$ and $F \subseteq F \cup T'$, it supports the same clauses w.r.t. $(F \cup T')$ with only variables of $H_i(F \cup T')$. Hence if it remains in $H'_{i+1}(F)$ it will also remain in $H_{i+1}(F \cup T')$. ■

**Claim 5.10.** *$Pr[T' \subset I$ and $V(T) \cap H = \emptyset] \leq Pr[T' \subset I]Pr[J \cap H' = \emptyset]$.*
**Proof.**

$$Pr[T' \subset I \text{ and } V(T) \cap H = \emptyset] \underbrace{\leq}_{J \subseteq V(T)} Pr[T' \subset I \text{ and } J \cap H = \emptyset] = Pr[J \cap H = \emptyset | T' \subset I]Pr[T' \subset I].$$

Hence, it is enough to show that $Pr[J \cap H = \emptyset | T' \subset I] \leq Pr[J \cap H' = \emptyset]$.

$$Pr[J \cap H' = \emptyset] = \sum_{F: J \cap H'(F) = \emptyset} Pr[I = F] \underbrace{\geq}_{H'(F) \subseteq H(F \cup T')} \sum_{F: J \cap H(F \cup T') = \emptyset} Pr[I = F] =$$

*break each set of clauses $F$ into $F' = F \setminus T'$ and $F'' = F \cap T'$*

$$\sum_{F': F' \cap T' = \emptyset, J \cap H(F' \cup T') = \emptyset} \sum_{F'': F'' \subseteq T'} Pr[I \setminus T' = F' \wedge I \cap T' = F''] =$$

*since the two sets of clauses are disjoint, and clauses are chosen independently*

$$\sum_{F': F' \cap T' = \emptyset, J \cap H(F' \cup T') = \emptyset} \sum_{F'': F'' \subseteq T'} Pr[I \setminus T' = F']Pr[I \cap T' = F''] =$$

$$\sum_{F': F' \cap T' = \emptyset, J \cap H(F' \cup T') = \emptyset} Pr[I \setminus T' = F'] \underbrace{\sum_{F'': F'' \subseteq T'} Pr[I \cap T' = F'']}_{1} =$$

$$\sum_{F': F' \cap T' = \emptyset, J \cap H(F' \cup T') = \emptyset} Pr[I \setminus T' = F'] =$$

*since $(I \setminus T') \cap T' = \emptyset$, and clauses are chosen independently, the event $\{T' \subseteq I\}$ is independent of the event $\{I \setminus T' = F'\}$.*

$$\sum_{F': F' \cap T' = \emptyset, J \cap H(F' \cup T') = \emptyset} Pr[I \setminus T' = F' | T' \subseteq I] = Pr[J \cap H = \emptyset | T' \subseteq I].$$

■