A Survey of Lower Bounds for Random 3-SAT via Differential Equations

Dimitris Achlioptas*
Microsoft Research

Abstract

It is widely believed that the satisfiability probability of random k-SAT formulae exhibits a sharp threshold as a function of the clauses-to-variables ratio. For the most studied case, $k = 3$, there have been a number of results during the last decade showing almost certain (un)satisfiability for specific values of this ratio, thus providing upper and lower bounds for the threshold's potential location. All lower bounds in this vein have been algorithmic, i.e. in each case a particular algorithm is shown to satisfy random instances of 3-SAT with probability $1-o(1)$ if their ratio of clauses to variables is below a certain value. Here we show how differential equations can serve as a generic tool for analyzing such algorithms, by rederiving most of the known lower bounds for random 3-SAT in a simple, uniform manner.

1 Introduction

It is widely believed that the probability of satisfiability for random k-SAT formulae exhibits a sharp threshold as the ratio of clauses to variables is increased. More precisely, let $F_k(n,m)$ denote a random formula in Conjunctive Normal Form with m clauses over n Boolean variables, where the clauses are chosen uniformly, independently and with replacement among all $2^k\binom{n}{k}$ non-trivial clauses of length k, i.e. clauses with k distinct, non-complementary literals[1]. Let

$$S_k(n,r) = \Pr[F_k(n,rn) \text{ is satisfiable}] \ .$$

In [14], the following possibility was put forward and has since become a folklore conjecture.

Satisfiability Threshold Conjecture *For every $k \geq 2$, there exists a constant r_k such that for all $\epsilon > 0$,*

$$\lim_{n\to\infty} S_k(n, r_k - \epsilon) = 1, \quad \textit{and} \quad \lim_{n\to\infty} S_k(n, r_k + \epsilon) = 0 \ .$$

For $k = 2$, Chvátal and Reed [14], Goerdt [23] and Fernandez de la Vega [18] independently proved the conjecture and determined $r_2 = 1$. Recall that 2-SAT being solvable in polynomial time [15] means that we have a *simple* characterization of unsatisfiable 2-SAT formulae. Indeed, [14, 23] make full use of this characterization as they proceed by focusing on the emergence of the "most likely" unsatisfiable subformulae in $F_2(n,rn)$. Also using this characterization, Bollobás et

*Microsoft Corporation, One Microsoft Way, Redmond, WA 98052. optas@microsoft.com

[1]While we adopt the $F_k(n,m)$ model throughout the paper, it is worth noting that all of the results we discuss hold in all standard models for random k-SAT, e.g. when clause replacement is not allowed and/or when each k-clause is formed by selecting k literals uniformly at random with replacement.

al. [10] recently determined the "scaling window" for random 2-SAT, showing that the transition from satisfiability to unsatisfiability occurs for $m = n + \lambda n^{2/3}$ as λ goes from $-\infty$ to $+\infty$.

For $k \geq 3$, much less progress has been made. Not only is the location of the threshold unknown, but even the existence of r_k has not been established. A big step towards the latter was made by Friedgut [21] who showed the existence of a sharp threshold around some critical *sequence of values*.

Theorem 1 ([21]) *For every $k \geq 2$, there exists a sequence $r_k(n)$ such that for all $\epsilon > 0$,*

$$\lim_{n\to\infty} S_k(n, r_k(n) - \epsilon) = 1, \quad \textit{and} \quad \lim_{n\to\infty} S_k(n, r_k(n) + \epsilon) = 0 \ .$$

While it is widely believed that the sequence $r_k(n)$ above converges, a proof remains elusive. We will find the following immediate corollary of Theorem 1 very useful.

Corollary 2 *If r is such that $\liminf_{n\to\infty} S_k(n,r) > 0$ then for any $\epsilon > 0$, $\lim_{n\to\infty} S_k(n, r - \epsilon) = 1$.*

Let us say that a sequence of events $\mathcal{E}_n$ holds *with high probability* (w.h.p.) if $\lim_{n\to\infty} \Pr[\mathcal{E}_n] = 1$. Let us say that $\mathcal{E}_n$ holds *with positive probability* if $\liminf_{n\to\infty} \Pr[\mathcal{E}_n] > 0$. In the rest of the paper, to simplify notation, we will write $r_k \geq r^*$ to denote that for $r < r^*$, $F_k(n, rn)$ is satisfiable w.h.p. (analogously for $r_k \leq r^*$). In these terms, we see that Corollary 2 above allows one to establish $r_k \geq r$ by only showing that $F_k(n, rn)$ is satisfiable with positive probability.

Even before the satisfiability threshold conjecture was stated, it was known that for all $k \geq 3$, $c_1(2^k/k) \leq r_k \leq c_2 2^k$, where c_1, c_2 are constants independent of k. The upper bound follows by observing that the expected number of satisfying truth assignments of $F_k(n, rn)$ is $o(1)$ for $r > 2^k$ [20]. The lower bound comes from analyzing the UNIT CLAUSE algorithm [13] and we will elaborate on it in Section 4. Despite significant efforts these bounds remain the best known for general k, up to the value of c_1, c_2, leaving a gap of order k. Closing this gap appears to be a challenging problem, closely related to hypergraph 2-colorability (Property B) [17, 2]. For general algorithmic results related to random formulae, see the paper by Franco [19] in this issue.

For the case $k = 3$, the first computationally non-trivial case, several upper and lower bounds have allowed us to close in, somewhat, on the potential value of r_3 (experimental evidence suggests $r_3 \approx 4.2$). Figure 1 summarizes the current state of the art. All upper bounds have been proved via probabilistic counting arguments (see the survey by Dubois, Kirousis and Stamatiou [16] in this volume). On the other hand, all lower bounds are algorithmic, i.e. in each case a particular algorithm is shown to satisfy $F_3(n, rn)$ w.h.p. for r below a certain value r^*. Note that in light of Corollary 2, showing positive probability of success for $r < r^*$ in fact suffices to establish $r_3 \geq r^*$.

The first lower bound for r_3 was given by Broder, Frieze and Upfal [11] who considered the *pure literal* heuristic on $F_3(n, rn)$. This heuristic satisfies a literal only if its complement does not appear in the formula, thus making only "safe" steps. They showed that for $r \leq 1.63$, w.h.p. the pure literal heuristic eventually sets all the variables implying $r_3 \geq 1.63$ (they also showed that for $r > 1.7$ w.h.p. the pure literal heuristic does not set all the variables).

Before the pure literal heuristic gave the first lower bound $r_3 > 1.63$, in [12] Chao and Franco had shown that the UNIT CLAUSE (UC) algorithm has positive probability of finding a satisfying truth assignment for $r < 8/3 = 2.66...$ They also showed that when UC is combined with a "majority" rule it succeeds with positive probability for $r < 2.9$. Since each of these algorithms succeeds only with positive probability, instead of w.h.p., these results did not imply $r_3 \geq 2.9$. Yet, by now, Corollary 2 allows us to immediately convert each of these results to a lower bound for r_3. Moreover, the style of analysis in [12] influenced a number of later papers [13, 14, 22, 5, 3, 4, 1]. To improve upon the bound provided by the pure literal heuristic, Frieze and Suen [22] considered two generalizations of

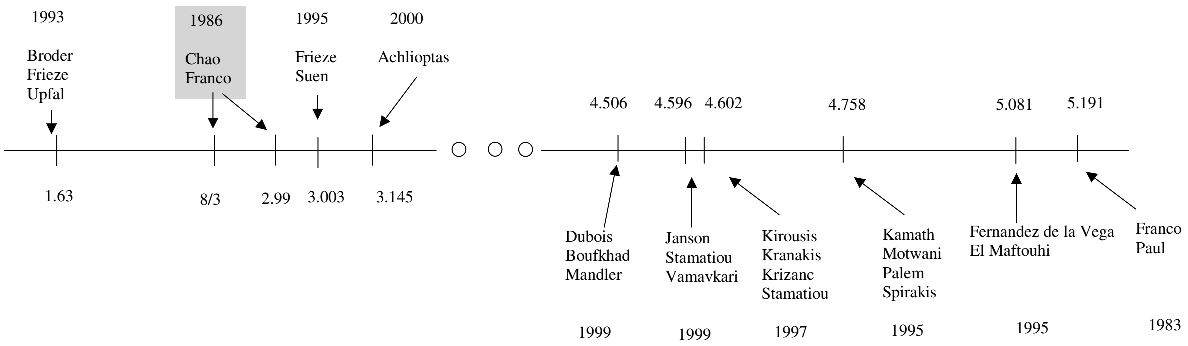

Figure 1: Known lower and upper bounds for r_3

UC, called SC and GUC respectively, and determined their exact probability of success on $F_3(n, rn)$. In particular, they showed that both heuristics succeed with positive probability for $r < 3.003...$ and fail w.h.p. for $r > 3.003...$ They further proved that a modified version of GUC, which performs a very limited form of backtracking, succeeds w.h.p. for $r < 3.003..$ thus yielding $r_3 \geq 3.003...$

In a recent paper, Achlioptas [1] showed that UC and all its variations discussed above [12, 14, 22] can be expressed in a common framework and analyzed uniformly by using differential equations. Specifically, after modeling each algorithm's execution as a Markov chain, one can apply a theorem of Wormald [32] to yield the following: the Markov chain corresponding to the execution of each algorithm on $F_3(n, rn)$ w.h.p. stays "near" a (deterministic) trajectory that can be expressed as the solution of a system of differential equations. At this powerful algebraic level, analyzing a new algorithm that falls within the framework is essentially reduced to calculating the *expected* conditional change of a small number of parameters of the Markov chain during a *single* step. Such a new satisfiability heuristic that sets two variables "at a time" was introduced and analyzed in [1], yielding $r_3 > 3.145$.

In this paper, we focus on presenting the framework and the techniques of [1] at a level that conveys the key ideas and the intuition behind them, without being bogged down by tedious technical details. We hope that, in the end, the reader will also feel that indeed only such details have been omitted. The material is organized along the lines of a lecture on the topic whose aim is to derive the known results in a simple and uniform fashion. Rather than presenting a comprehensive (but unmotivated and perhaps hard to digest) framework up front, we have chosen to introduce each new idea at the point where it first becomes useful in analyzing some algorithm.

2 The Framework

All satisfiability algorithms that have been analyzed on random formulae share two characteristics. Both characteristics are motivated by the need to address a common problem: controlling the conditioning imposed on future steps of the algorithm by the steps made up to the current point.

The first such characteristic is that all algorithms are backtrack-free, i.e. once a variable is set to TRUE (FALSE), its value is never later changed to FALSE (TRUE) in the course of the algorithm. The backtracking present in the algorithm of Frieze and Suen [22] escapes this paradigm only superficially. In particular, w.h.p. no more than polylog(n) value assignments are ever reversed and, moreover, any such reversal occurs for simple "local" reasons. Thus, this backtracking is in no way similar to the "search space exploration" implicit in almost all satisfiability algorithms used in practice. Unfortunately, the lack of a simple dependence structure between decisions in such algorithms makes their analysis (appear) beyond the reach of current mathematical techniques.

The second shared characteristic of all the algorithms analyzed so far is that, in order to preserve independence, they are limited to following relatively simple rules in order to determine which variable(s) to set in each step and what value(s) to assign. More precisely, these rules are such that they could be followed even if one had a very limited form of access to the input formula (thus leading to a correspondingly simple form of conditional independence). Perhaps the best way to make this last idea concrete is by introducing a simple game.

2.1 The Card Game

All the algorithms that we will consider make n iterations, permanently setting one variable in each iteration. As soon as a clause is satisfied it is removed from the formula, never to be considered again. Similarly, as soon as a literal in an i-clause c becomes dissatisfied, that literal is removed from c which is now considered an $(i-1)$-clause (we will say that c "shrunk").

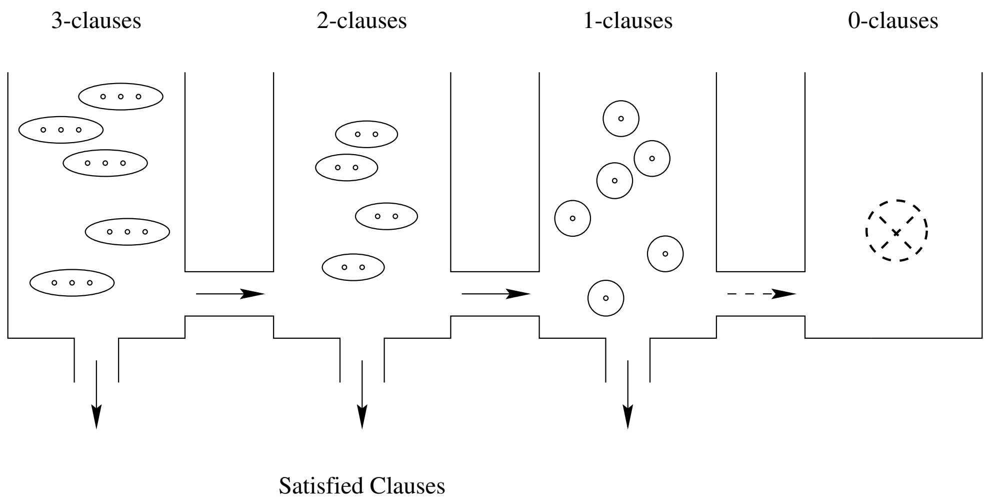

Figure 2: Evolution of clauses in card-game algorithms

Thus, the question is: *which* variable gets set in each step, and *how*? For all algorithms, except for the pure literal heuristic, these decisions can be made in the context of the following game.

Imagine representing a k-SAT formula by a column of k cards for each k-clause, each card bearing the name of one literal. Assume, further, that originally all the cards are "face-down", i.e. the literal on each card is concealed (and we never had an opportunity to see which literal is on each card). At the same time, assume that an intermediary knows precisely which literal is on each card, and to interact with the intermediary we are allowed to either

- Point to a particular card, or,
- Name a variable that has not yet been assigned a value.

In response, if the card we point to carries literal ℓ, then the intermediary reveals (flips) all the cards carrying $\ell, \bar{\ell}$. Similarly, if we name variable v, the intermediary reveals all the cards carrying $v, \bar{v}$. In either case, faced with all the occurrences of the chosen variable we (somehow) proceed to decide which value to assign to it. Having done so, we remove all the cards corresponding to dissatisfied literals and all the cards (some of them still concealed) corresponding to satisfied clauses. As a result, at the end of each step only "face-down" cards remain, containing only literals corresponding to unset variables. Thus, certain "formalities" notwithstanding, the card game brings us the key property of uniform randomness (Lemma 3 below). Before stating Lemma 3 let us introduce some notation that we will use throughout the paper.

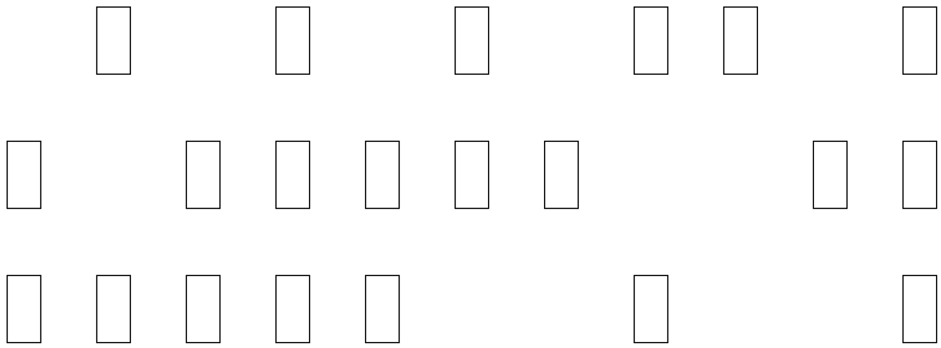

Figure 3: A typical configuration of the card game

Notation For any literal ℓ, $\mathrm{var}(\ell)$ will denote its underlying variable. For any set V of Boolean variables, $L(V)$ will denote the set of $2|V|$ literals on the variables of V. "At time t" will mean after t iterations of each algorithm have been performed, i.e. after t variables have been set.

$\mathcal{V}(t)$ will denote the random *set* of variables not assigned a truth value at time t.

$\mathcal{S}_i(t)$ will denote the random *set* of i-clauses remaining at time t.

$C_i(t) \equiv |\mathcal{S}_i(t)|$ will denote the *number* of i-clauses remaining at time t.

Lemma 3 (Uniform randomness) *For every $0 \leq t \leq n$, conditional on all information exposed thus far, each clause in $\mathcal{S}_i(t)$ contains a uniformly random set of i distinct, non-complementary literals from $L(\mathcal{V}(t))$. Thus, if $\mathcal{V}(t) = X$ and $C_i(t) = q_i$, the set of i-clauses remaining at time t form a random i-SAT formula $F_i(|X|, q_i)$ on the variables in X.*

It is rather easy to see why the pure literal heuristic cannot be expressed via the card game: unless only one clause remains, Figure 3 does not convey sufficient information for us to be able to "pick" a pure literal. On the other hand, clause-length information *is* available and it can be exploited by pointing at certain cards. In particular, since the algorithms never backtrack, whenever there are clauses (columns) with only one literal (card) left, it is prudent to point at any one of these cards right away and satisfy the underlying literal. It is intuitively clear that postponing such action can only hurt the probability of finding a satisfying truth assignment. (If a 0-clause is generated while doing this, then clearly the algorithm fails; for the sake of uniformity, though, we will assume that the algorithm continues unaffected until all variables are set.) We will sometimes call clauses of length 1 *unit clauses* and the corresponding steps in which we satisfy them *forced*.

From the above, we already see that we can only have a potentially interesting strategy when the residual formula does not contain unit clauses. We will call such steps *free*. In terms of *which* variable to set, in a free step it will turn out that each algorithm will either completely ignore clause-length information (by naming a random unset variable) or it will pick a variable at random among those appearing in clauses of a certain length (by pointing at a random card appearing in a random column with the specified number of cards). In terms of *how* the chosen variable is set we will see much greater variation between algorithms, giving rise to a number of interesting tradeoffs.

3 The UNIT CLAUSE Algorithm

Perhaps the simplest possible algorithm expressible via the card game is the UNIT CLAUSE (UC) algorithm, introduced and analyzed by Chao and Franco [12].

UNIT CLAUSE

```
1. If there are any 1-clauses (forced step)
     Pick a 1-clause uniformly at random and satisfy it

2. Otherwise (free step)
     Pick an unset variable uniformly at random and
     assign it 0/1 uniformly at random
```

Clearly, at each step of type 1 UC might generate a clause of length 0 (contradiction) and such a clause will never be removed. On the other hand, if this never happens then UC finds a satisfying truth assignment, in which case we say that it *succeeds*. As we mentioned earlier, for the sake of the analysis, it will be simpler to always let the algorithm continue until all variables are set, even if a 0-clause was generated at some point. This allows us to analyze each step without having to consider any conditioning implied by the fact that the step actually takes place.

To analyze UC let us first note that if A is any algorithm expressible in the card game then A fails w.h.p. if at any point during its execution the density of the underlying 2-SAT subformula exceeds 1. That is, if for some $\delta, \epsilon > 0$ we have $C_2(t_b) > (1+\delta)(n-t_b)$ where $t_b \leq (1-\epsilon)n$, then w.h.p. the underlying 2-SAT subformula at time t_b is unsatisfiable; in that case, since A never backtracks, a 0-clause will certainly be generated at some $t \geq t_b$. What might be somewhat surprising at first glance is that this necessary condition is actually sufficient. Below we give an intuitive explanation as to why this is true for UC, followed by a precise statement of the condition (Lemma 4) that we will use in the analysis of all algorithms.

Intuition: As we will see shortly, for all $t = 0, \ldots, n-1$, the expected number of unit clauses generated in step t of UC is $C_2(t)/(n-t)$, i.e. it is equal to the density of the underlying 2-SAT formula at the time. As a result, as long as this density is bounded by $1-\delta$ for some $\delta > 0$, since UC can satisfy (and thus remove) one unit clause in each step, unit clauses will not accumulate. In fact, under this condition, $C_1(t)$ behaves very much like the queue size in a stable server system. In particular, there exists a constant $M = M(\delta)$ such that w.h.p. $\sum_{t=0}^{n-1} C_1(t) < Mn$.

Observe now that if $\mathcal{S}_1(t) = \emptyset$, then the probability that a 0-clause is generated in step t is 0 since every clause loses at most one literal in each step. Further, if $\mathcal{S}_1(t) \neq \emptyset$ then in step t UC picks some $\ell \in \mathcal{S}_1(t)$ at random and satisfies it. Clearly, if $\overline{\ell} \notin \mathcal{S}_1(t)$ then no 0-clause will be generated in step t. In the card game, picking ℓ corresponds to picking randomly a column with one card and pointing to the unique card in it. Note now that if at that point we were to flip only the card pointed at, we would still have no information whatsoever regarding which literal is on any other card. Hence, by uniform randomness, we get that conditional on $C_1(t) = a > 0$, the probability that no 0-clause is generated in step t is

$$\left(1 - \frac{1}{2(n-t)}\right)^{a-1} \quad . \tag{1}$$

To complete the argument, let us fix a small $\epsilon > 0$ and take $t_e = \lfloor (1-\epsilon)n \rfloor$. Assume now that for a given r we can prove that there exists $\delta > 0$ such that w.h.p. $C_2(t) < (1-\delta)(n-t)$ for all $0 \le t \le t_e$. Then, using our discussion above, it is not hard to show that in this case $\Pr[\mathcal{S}_0(t_e) = \emptyset] > \exp(-M/\epsilon)$. In other words, if w.h.p. during the first t_e steps the density of the 2-clauses is uniformly bounded away from 1, then with positive probability the algorithm does not fail during those t_e steps. (The reason for introducing t_e is to get a non-trivial lower bound, i.e. $\Omega(n)$, for the denominator appearing in (1)). As we will see, dealing with the last $n - t_e$ variables will be straightforward since at that point the remaining formula will be "very easy". In particular, it can be dealt with by using an argument that is much easier to analyze than analyzing the last $n - t_e$ steps of each algorithm.

Lemma 4 *Let A be any algorithm expressible in the card game with the property that it always satisfies some unit clause whenever unit clauses exist. If $\delta, \epsilon > 0$ and t_e are such that $t_e \le (1-\epsilon)n$ and*

$$w.h.p. \quad C_2(t) < (1-\delta)(n-t) \quad \textit{for all } 0 \le t \le t_e \ ,$$

then, there exists $\rho = \rho(\delta, \epsilon) > 0$ such that $\Pr[\mathcal{S}_0(t_e) \cup \mathcal{S}_1(t_e) = \emptyset] > \rho$.

Note that Lemma 4 asserts that, in fact, no unit clauses will exist either as at time t_e; this will help simplify the analysis. At least intuitively, i.e. if we accept that $C_1(t)$ behaves likes the queue in a stable server system, this assertion parallels the well-known fact that in any fixed step the server is idle with constant probability.

4 Tracing the number of 2-clauses

In view of Lemma 4, we see that to analyze the performance of UC on $F_3(n, rn)$ we need to trace the evolution of the number of 2-clauses (and hence of the 2-clause density) during the algorithm's execution. To do this, we are going to setup some machinery which will allow us to approximate the evolution of 2-clauses and 3-clauses via a system of differential equations. The idea of using differential equations to approximate discrete random processes goes back at least to Kurtz [27, 28]. It was first applied in the analysis of algorithms by Karp and Sipser [25]. The astute reader might object that using such machinery is overkill for an algorithm as simple as UC where one can derive the desired results by elementary combinatorial means. Indeed, we chose to introduce the machinery in this most straightforward setting to minimize "peripheral" distractions.

Let $\vec{U}(t) = \langle C_2(t), C_3(t) \rangle$ be a vector describing the number of 2- and 3-clauses at time t and let $\mathbf{H}(t) = \langle \vec{U}(0), \ldots, \vec{U}(t) \rangle$ be a $2 \times (t+1)$ matrix describing the entire history of the number of 2-clauses and 3-clauses up to time t. For random variables X, Y let us write $X \overset{D}{=} Y$ if for every value x in the domain of X, $\Pr[X = x] = \Pr[Y = x]$. Finally, let $\mathrm{Bin}(N, s)$ denote the Binomial random variable with N trials each having probability of success s.

Lemma 5 *Let $\Delta C_i(t) = C_i(t+1) - C_i(t)$. For all $0 \le t \le n-3$, conditional on $\mathbf{H}(t)$,*

$$\Delta C_3(t) = -X \ , \tag{2}$$
$$\Delta C_2(t) = Y - Z \ , \tag{3}$$

where

$$X \stackrel{D}{=} \text{Bin}\left(C_3(t), \frac{3}{n-t}\right) \quad , \quad Y \stackrel{D}{=} \text{Bin}\left(C_3(t), \frac{3}{2(n-t)}\right) \quad , \quad Z \stackrel{D}{=} \text{Bin}\left(C_2(t), \frac{2}{n-t}\right) \quad . \tag{4}$$

Proof. Intuitively, each negative term in (2), (3) represents the number of clauses leaving $\mathcal{S}_i(t)$ during step t, either as satisfied or as shrunk, while the positive term expresses the fact that the clauses leaving $\mathcal{S}_3(t)$, with probability 1/2 move to $\mathcal{S}_2(t+1)$. To prove the lemma we first claim that for every $0 \le t \le n-1$ the literal satisfied during step t is chosen uniformly at random among the literals in $L(\mathcal{V}(t))$. To prove this, we simply note that when $\mathcal{S}_1(t) \neq \emptyset$ the claim follows by Lemma 3 applied to $\mathcal{S}_1(t)$, while when $\mathcal{S}_1(t) = \emptyset$ it follows from the definition of the algorithm.

Let ℓ be the literal chosen to be satisfied during step t. As ℓ is uniformly random among the literals in $L(\mathcal{V}(t))$, by Lemma 3, we know that every clause $c \in \mathcal{S}_i(t)$, $i = 2, 3$, contains $\text{var}(\ell)$, independently of all other clauses and with the same probability. As there are $n-t$ unset variables, if c has i literals then this probability is $i/(n-t)$. This yields the negative terms in (2),(3) as each clause containing $\text{var}(\ell)$ is removed from the set to which it belonged at time t. To get the positive term we note that as ℓ is uniformly random, by Lemma 3, if $c \in \mathcal{S}_3(t)$ contains one of $\ell, \bar{\ell}$ then it contains $\bar{\ell}$ with probability 1/2. □

The difference equations (2),(3) above, suggest a "mean path" for C_3, C_2, i.e. a path corresponding to the trajectories these random variables would follow if in each step their actual change was equal to its conditional expectation. Clearly, in general, there is no reason to believe that arbitrary random variables remain anywhere close to their mean path. Here, though, there are two reasons suggesting that w.h.p. this will indeed be the case for all $t \le t_e = (1-\epsilon)n$, where $\epsilon > 0$ is an arbitrarily small constant.

The first such reason is that the number of clauses involved in each "flow" of Figure 2 is the result of an experiment with $O(n)$ trials (one for each clause in the originating bucket), each having probability of success $O(n^{-1})$ (the assumption $t \le t_e$ is important here). As a result, the probability that more than s clauses enter or leave each $\mathcal{S}_i$ in a given step is roughly $e^{-\beta s}$, for some $\beta > 0$; in particular, there is $\gamma > 0$, such that w.h.p. for all $t \le t_b$, $|\mathcal{S}_i(t+1) \setminus \mathcal{S}_i(t)| < \gamma \log n$, for $0 \le i \le 3$.

The second, and perhaps more subtle, reason is that for $t \le t_e$, the expectation of $\Delta C_i(t)$, $i = 2, 3$, is a "smooth" function, both with respect to t and with respect to $C_i(t)$, $i = 2, 3$. That is, changing any of $t, C_2(t), C_3(t)$ by $o(n)$ affects each $\Delta C_i(t)$ by $o(1)$. Hence, even if the process has deviated by $o(n)$ from the mean path, its dynamics are not significantly affected. Note that $C_1(t)$, for example, does not enjoy this property: knowing whether $\mathcal{S}_1(t) \neq \emptyset$ or not affects $\Delta C_1(t)$ by 1.

All in all, there are two key properties that the dynamics of C_i, $i = 2, 3$, enjoy: i) the conditional change in each step has constant expectation and strong tail bounds, and ii) knowing the parameters of the process, i.e. $t, C_2(t), C_3(t)$, within $o(n)$ suffices to determine $\Delta C_i(t)$ within $o(1)$. It is precisely these two properties that will allow us to use a theorem of Wormald [32], Theorem 8 in Appendix A, to approximate the evolution of $C_i(t)$, $i = 2, 3$.

While the framework of Theorem 8 might appear rather technical at first glance, the spirit of the theorem is not hard to describe. We have k random variables, where k is arbitrary but fixed, that evolve jointly over a period of $O(n)$ steps and which are guaranteed to never exceed Bn for some constant B. Further, each of these random variables has the following property: for any possible history of the joint process, the conditional probability that the single-step change deviates significantly from its conditional expectation is small. Moreover, the dynamics of each random variable are "smooth", a notion captured in the continuous, deterministic setting. To pass to that setting we consider a scaled version of the state-space of the process, resulting by dividing

each parameter by n. This defines a corresponding region in $\mathbb{R}^{k+1}$ (since time is also a parameter) in which smoothness of the process will be considered. In particular, and this is the more technical part of the theorem, assume that there exists a region in $\mathbb{R}^{k+1}$ (the domain D in Theorem 8) inside which the (scaled version of the) process is well-behaved. That is, there exist k "smooth" functions defined on D (satisfying a Lipschitz condition), which when given the scaled parameters of the process, return the conditional expected change of each of the k random variables within $o(1)$. The theorem then asserts that treating these k functions as derivatives and solving the corresponding system of differential equations yields a set of k functions (trajectories) that w.h.p. approximate the (scaled) actual random process well (within $o(n)$) for (at least) as long as these trajectories stay inside D.
For C_3, C_2 the parallels between the probabilistic and the deterministic domain are drawn below.

$$\mathbf{E}(\Delta C_3(t) \mid \mathbf{H}(t)) = -\frac{3C_3(t)}{n-t} \qquad\Bigg|\qquad \frac{dc_3}{dx} = -\frac{3c_3(x)}{(1-x)} \qquad [x \equiv t/n]$$

$$C_3(0) = rn \qquad\Bigg|\qquad c_3(0) = r$$

$$\mathbf{E}(\Delta C_2(t) \mid \mathbf{H}(t)) = \frac{1}{2}\times\frac{3C_3(t)}{n-t} - \frac{2C_2(t)}{n-t} \qquad\Bigg|\qquad \frac{dc_2}{dx} = \frac{3c_3(x)}{2(1-x)} - \frac{2c_2(x)}{1-x}$$

$$C_2(0) = 0 \qquad\Bigg|\qquad c_2(0) = 0$$

Solving the two differential equations above and taking into account the initial conditions we get $c_3(x) = (1-x)^3$, $c_2(x) = \frac{3}{2}rx(1-x)^2$. Thus, applying Theorem 8 yields

Lemma 6 *Fix $\epsilon > 0$ and let $t_e = \lfloor (1-\epsilon)n \rfloor$. If* UC *is applied to $F_3(n, rn)$ then w.h.p. for $0 \le t \le t_e$,*

$$C_i(t) = c_i(t/n) \cdot n + o(n) \ , \tag{5}$$

where $c_3(x) = r(1-x)^3$ and $c_2(x) = \frac{3}{2}rx(1-x)^2$.

Proof. By Lemma 5, we can apply the Chernoff bound to bound $\Pr[\Delta C_i(t) > n^{1/5} \mid \mathbf{H}(t)]$, for each $i = 2, 3$. Thus, for any $\epsilon > 0$ the lemma follows by applying Theorem 8 with $k = 2$, $Y_i(t) = C_{i+1}(t)$, $B = r$, $m = n-3$, $f_1(s, z_1, z_2) = \frac{3z_2}{2(1-s)} - \frac{2z_2}{1-s}$, $f_2(s, z_1, z_2) = -\frac{3z_2}{2(1-s)}$ and D defined by $-\epsilon < s < 1$, $-\epsilon < z_i < r$, for $i = 1, 2$. □

Given Lemma 6, we are now ready to determine the values of r for which UC succeeds. First, recall that the key quantity is the density of the 2-clauses $C_2(t)/(n-t)$ which in the deterministic domain corresponds to $c_2(x)/(1-x) = \frac{3}{2}rx(1-x) \le 3r/8$, the inequality being tight for $x = 1/2$.

– For any $\delta > 0$, if $r = (8/3)(1+\delta)$ and $t_b = \lfloor n/2 \rfloor$ then Lemma 6 yields that w.h.p. $C_2(t_b) > (1+\delta/2)(n-t_b)$. Therefore, UC fails w.h.p. for such r.

– On the other hand, let us fix $\epsilon = 1/10$ yielding $t_e = \lfloor (9/10)n \rfloor$ and take $r = (8/3)(1-\delta)$, where $\delta > 0$ is any constant. By Lemma 6 we have that w.h.p. for all $0 \le t \le t_e$, $C_2(t) < (1-\delta/2)(n-t)$. Lemma 4 then implies that with positive probability there are no empty clauses or unit clauses at time t_e, i.e. $\mathcal{S}_0(t_e) \cup \mathcal{S}_1(t_e) = \emptyset$. Furthermore, Lemma 6 asserts that $C_3(t_e) = r(1-9/10)^3 n + o(n)$ and $C_2(t_e) = \frac{3}{2}r(9/10)(1-9/10)^2 n + o(n)$ implying that w.h.p. $C_3(t_e) + C_2(t_e) < (3/4)(n-t_e)$. Now, to conclude the proof, we argue as follows. Given a formula $F_3(n, rn)$ we will run UC for exactly t_e

steps and then remove one random literal from each remaining 3-clause. If $r = (8/3)(1-\delta)$, with positive probability we will be left with a random 2-SAT formula with $n - t_e = \Omega(n)$ variables and fewer than $(3/4)(n - t_e)$ clauses; such a formula is satisfiable w.h.p. Thus, our original formula was satisfiable with positive probability and as $\delta > 0$ was arbitrary, Corollary 2 implies $r_3 \geq 8/3$.

The proof structure that we used in this section to establish $r_3 \geq 8/3$ actually works for all the algorithms that we will see in the remaining sections. Thus, in each such section we will mostly focus on analyzing the dynamics of $C_2(t), C_3(t)$ in order to get an analogue of Lemma 6. The corresponding lower bound for r_3 will always follow by i) making sure $c_2(x) < (1-x)$ for $x \in [0,1]$, and ii) providing an appropriate t_e at which the remaining formula is "easy".

Similarly, one can analyze UC for general k. The resulting differential equations are

$$\begin{aligned}
\frac{dc_k}{dx} &= -\frac{kc_k(x)}{1-x} , & c_k(0) = r \\
&\vdots \\
\frac{dc_i}{dx} &= \frac{1}{2} \times \frac{(i+1)c_{i+i}(x)}{1-x} - \frac{ic_i(x)}{1-x} , & c_i(0) = 0 \\
&\vdots
\end{aligned}$$

yielding that for $i \geq 2$,

$$c_i(x) = r \binom{k}{i} 2^{i-k} x^{k-i} (1-x)^i .$$

Requiring $c_2(x)/(1-x) < 1$ now yields $r < \alpha 2^k/k$, where $\alpha = 1/2 \left(\frac{k-1}{k-2}\right)^{k-2} \to e/2$. In fact, one can show that if $c_3(x)/(1-x) < 2/3$, then UC succeeds w.h.p. (yielding a lower bound for r_k without invoking Corollary 2). This indeed holds for all $r < \alpha' 2^k/k$, where $\alpha' \to e^2/8$.

5 UC with majority

As Lemma 4 suggests, for any algorithm expressible in the card game the key to success lies in keeping the density of 2-clauses bounded away from 1. As a result, a natural goal for any such algorithm is to keep the number of 2-clauses as small as possible. While such a goal cannot be served during forced steps, one can improve over UC by attempting to minimize the number of 3-clauses that become 2-clauses, whenever possible. This is precisely what UC with majority (UCWM) does. This algorithm was introduced and analyzed in [12] along with UC.

<u>UNIT CLAUSE WITH MAJORITY</u>

1. `If there are any 1-clauses`
   `Pick a 1-clause uniformly at random and satisfy it`

2. `Otherwise`

   (a) `Pick an unset variable` x `uniformly at random`

   (b) `If` x `appears positively in at least half the remaining 3-clauses`
   `Set` $x =$ TRUE

```
(c) Otherwise
        Set x = FALSE
```

In terms of Figure 2 UCWM amounts to the following. In UC each total flow out of a bucket was divided evenly between satisfied and shrunk clauses. Here, during free steps, the algorithm biases the flow out of bucket 3 so that more clauses end up satisfied than in bucket 2 (all remaining flows remain evenly split). This curtails the rate of growth of $C_2(t)$, allowing the algorithm to handle denser formulae than UC. (We will discuss the 2-clauses "right to vote" in Section 7.)

In terms of analyzing UCWM the first thing to note is that the dynamics of $C_3(t)$ are identical to those under UC. In particular, similarly to UC, for every $0 \le t \le n-3$ and for every $c \in \mathcal{S}_3(t)$, the probability that the variable set in step t appears in c is $3/(n-t)$. This is because the choice of that variable is either random, in free steps, or is mandated by a unit clause, in which case the claim follows from uniform randomness. Hence, to analyze the performance of UCWM we simply need to analyze the dynamics of $C_2(t)$.

Unfortunately, in attempting to apply Wormald's Theorem we now hit a snag. The dynamics of $C_2(t)$ under UCWM, unlike those under UC, depend on whether the t-th step is free or not, i.e. whether $\mathcal{S}_1(t) = \emptyset$. Note that this is desirable (a feature, not a bug) as otherwise we get UC and the whole idea was to improve over UC by exploiting free steps. However, keeping track of $C_1(t)$ in the context of the theorem is hopeless since determining the expected change of $C_1(t)$ in step t cannot be done while only knowing $C_1(t)$ within $o(n)$.

One approach to dealing with this issue is to consider the entire process at a "microscopic" level, determining the joint evolution of $C_i(t)$, $i = 1, 2, 3$ at a much finer resolution than what Theorem 8 requires. While this approach can be made to work in certain cases [22], it leads to numerous technical issues. Moreover, the complexity of these issues tends to grow along with the complexity of the underlying algorithm, imposing significant limitations on the nature of algorithms that can be analyzed in this manner. Luckily, similarly to Section 3, an intuition from queuing theory (see [26, 24]) will allow us to take a much easier way out.

5.1 The lazy-server lemma

In UCWM, similarly to UC, both the choice of which variable to set in step t and the value assigned to that variable is independent of $\mathcal{S}_2(t)$. As a result, the flow from bucket 2 to bucket 1 in step t, the $(2 \to 1)$-flow, is again distributed as $\mathrm{Bin}(C_2(t), 1/(n-t))$ and the rate at which 1-clauses are generated is $C_2(t)/(n-t)$. Now, viewing $C_1(t)$ as the queue of a server system, and assuming that $C_2(t)$ does not change too rapidly, we know that if $C_2(t)/(n-t) < 1$, then around time t the server is idle, i.e. $\mathcal{S}_1(t) = \emptyset$, a fraction $1 - C_2(t)/(n-t)$ of the time. While this intuition is correct, converting it to a probabilistic statement that we can use to determine the evolution of $C_2(t)$ is problematic; the logic we used is circular: the rate at which unit clauses are generated affects the probability that the algorithm takes a free step, which affects the value of $C_2(t)$, which in turn affects the rate at which unit clauses are generated. A cultural paradigm will come to our rescue.

Perhaps the simplest way to present the "lazy-server" idea is to first pass to the continuous setting by considering a server capable of handling w units of work per step, where w is some positive real number and work arrives at a rate of λ units per step from some distribution that does not have extreme variance. In this setting it is well-known that if $\lambda < w$, then the server's queue will remain bounded, which in our setting corresponds to the event $\sum_{t=0}^{n} C_1(t) < Mn$, for some constant M. Thus, if we only care that the queue remains bounded, then it is clear that we can replace the server of capacity w with a server of capacity $\lambda + \theta$, for some arbitrarily small

$\theta > 0$; while the average queue size will indeed increase as $\theta \to 0$, the queue will remain bounded for any fixed such $\theta > 0$. Moreover, and this is the key idea, we claim that we can implement this replacement in a randomized rather than a deterministic manner: we will keep the server of capacity w, but in each step have it "go to sleep" with probability $p = (\lambda + \theta)/w$.

To get a feel for the claim, note that the concentration of the Binomial random variable implies that in every reasonably long period of t steps the server will attempt to serve the queue very close to tp times. Also, precisely because the server is "lazy", almost every time (s)he attempts to serve the queue (s)he will find work there waiting. The main feature, though, is that now we have a process that is completely independent of the input stream and which, as $\theta \to 0$, tends to capture all of the server's idle time. That is, as $\theta \to 0$, the probability that the server attempts to serve the queue and finds it empty tends to 0. This "lazy-server" idea was introduced in [1] and its precise statement can be found in Appendix A (Lemma 9).

In our setting, the lazy-server idea will amount to the following. Rather than *always* satisfying some unit clause whenever such clauses exist, the algorithm will rather *attempt* to satisfy some unit clause in every step *with probability* $u = \min\{(1+\theta)C_2(t)/(n-t), 1\}$, for some small $\theta > 0$. If in such an attempt the algorithm finds that no unit clauses exist, then rather than doing something "clever" it will simply satisfy a randomly chosen literal (the reason for this will become clear below).

<u>UCWM WITH LAZY-SERVER POLICY</u>

1. For $t = 1, \ldots, n$
   (a) Determine $U(t)$ [Set $U(t) = 1$ with probability u]
   (b) If $U(t) = 1$
      i. If there are any 1-clauses
         Pick a 1-clause uniformly at random and satisfy it
      ii. Otherwise
         Pick an unset variable uniformly at random and
         assign it 0/1 uniformly at random
   (c) Otherwise
      i. Pick an unset variable x uniformly at random
      ii. If x appears positively in at least half the remaining 3-clauses
         Set $x =$ TRUE
      iii. Otherwise
         Set $x =$ FALSE

Recall that, by uniform randomness, in every step t satisfying a unit clause or a randomly chosen literal has exactly the same effect on the evolution of the 2-clauses. Therefore, by avoiding to do something clever whenever $U(t) = 1$ and $\mathcal{S}_1(t) = \emptyset$, we gain that the dynamics of the number of 2-clauses are *completely* independent of the presence of 1-clauses, depending only on the random coin flips that determine $U(t)$. Furthermore, the "wastefulness" of this approach (in terms of missed opportunities) tends to 0 as $\theta \to 0$. In particular, we can get the following analogue of Lemma 4.

Lemma 7 *Let A be any algorithm expressible in the card game which in every step t attempts to satisfy a unit clause with probability $u = u(t, C_2(t), C_3(t))$. If $\delta, \epsilon > 0$ and t_e are such that $t_e \leq (1-\epsilon)n$ and*

$$w.h.p. \quad C_2(t) < (1-\delta)(n-t) \times u \quad \textit{for all } 0 \leq t \leq t_e \ ,$$

then, there exists $\rho = \rho(\delta, \epsilon) > 0$ such that $\Pr[\mathcal{S}_0(t_e) \cup \mathcal{S}_1(t_e) = \emptyset] > \rho$.

Having dealt with the distinction between free and forced steps, we now need to understand the distribution of the number of 3-clauses that become 2-clauses in step t. In particular, as we saw earlier, if v is the variable set in step t then the number of 3-clauses containing $v, \overline{v}$ is distributed as $\mathrm{Bin}(C_3(t), 3/(n-t))$. Further, each such clause contains each of $v, \overline{v}$ with equal probability. Thus, if $U(t) = 1$ then each such clause becomes a 2-clause with probability $1/2$. On the other hand, if $U(t) = 0$ it is not hard to show that, asymptotically, the number of 3-clauses that becomes 2-clauses is distributed like the minimum of two independent Poisson random variables each with mean $\lambda = \frac{3C_3(t)}{2(n-t)}$. Unfortunately, it is not possible to get a closed-form expression for the expectation, $M(\lambda)$, of this last random variable. Nonetheless, if we let $P(\lambda; i) \equiv \Pr[\mathrm{Po}(\lambda) = i]$ then

$$B_q(\lambda) = \lambda - \sum_{j=0}^{q} \sum_{k=0}^{q} P(\lambda; j) P(\lambda; k) \left(\min\{j, k\} - \frac{j+k}{2} \right) \geq M(\lambda)$$

and the bound gets better as q is increased. In particular, taking $q = 40$ is manageable computationally and gives an excellent bound for $\lambda \leq 5$. In our setting, since $C_3(t) = r(1-t/n)^3 + o(n)$ and $\lambda = \frac{3C_3(t)}{2(n-t)}$ we see that if $r < 10/3$ then $\lambda \leq 5$ throughout the evolution of the process.

Since we only have a bound on the expected $(3 \to 2)$-flow during free steps, we somehow need to argue that we can use this bound in the context of Theorem 8 and still get a useful result for the number of 2-clauses. One way to do this would be to manipulate the differential equations and show that using an upper bound for $M(\lambda)$ yields an upper bound for the solution corresponding to the actual process. While this is possible here, we are going to instead use a trick that simplifies matters a lot, without sacrificing any sharpness in the analysis (other than the sacrifice inherent in that we are using just an upper bound). The trick is rather generic and it will be most useful in Section 7.1 where arguing by direct manipulation of the differential equations would be daunting. The idea is to "dumb down" our algorithm a little bit so that in fact the expected number of 3-clauses that become 2-clauses in a free step *equals* $B_q\left(\frac{3C_3(t)}{2(n-t)}\right)$. We do this as follows: whenever $U(t) = 0$ (free step) the algorithm tosses another coin, with appropriate probability, in order to decide if it will actually set v "by majority" or "at random". Since $B_q(\lambda)$ is an upper bound for $M(\lambda)$ for all λ such a choice of probability clearly exists. As a result we can now apply Wormald's Theorem for C_3, C_2 in this modified algorithm where we have exact expressions for the expected change in each step. Recalling that $\Pr[U(t) = 1] = \min\left\{1, \frac{(1+\theta)C_2(t)}{n-t}\right\}$ we have

$$\begin{aligned} &\mathbf{E}(C_2(t+1) - C_2(t) \mid \mathbf{H}(t)) = \\ &\quad \Pr[U(t) = 1] \times \left(\frac{3C_3(t)}{2(n-t)} - \frac{2C_2(t)}{n-t} \right) + (1 - \Pr[U(t) = 1]) \times \left(B_q\left(\frac{3C_3(t)}{2(n-t)}\right) - \frac{2C_2(t)}{n-t} \right) \ . \end{aligned}$$

Letting $v(x, c_2(x)) = \min\left\{1, \frac{(1+\theta)c_2(x)}{1-x}\right\}$ and recalling that $c_3(x) = r(1-x)^3$ this corresponds to

$$\frac{dc_2}{dx} \ = \ v(x, c_2(x)) \times \left(\frac{3c_3(x)}{2(1-x)} - \frac{2c_2(x)}{1-x} \right) + (1 - v(x, c_2(x))) \times \left(B_q\left(\frac{3c_3(x)}{2(1-x)}\right) - \frac{2c_2(x)}{1-x} \right)$$

$$= \quad v(x, c_2(x)) \times \left(\frac{3r(1-x)^2}{2} - \frac{2c_2(x)}{1-x}\right) + (1 - v(x, c_2(x))) \times \left(B_q\left(\frac{3r(1-x)^2}{2}\right) - \frac{2c_2(x)}{1-x}\right) \ .$$

Solving[2] the above differential equation, with $q = 40$, $\theta = 10^{-5}$ and $r = 3.001$ yields that $c_2(x) < (1 - 10^{-6})(1 - x)$ for all $x \in [0, 1]$; moreover we get $c_2(0.9) < 4/100$. Since $c_3(0.9) = r(1/10)^3 < 1/100$, an argument similar to the one used for UC yields $r_3 \geq 3.001$.[3]

6 Going on clause-length alone

In [22] two different extensions of UC were considered. The first one, GUC (GENERALIZED UNIT CLAUSE), always picks a shortest remaining clause and satisfies a random literal in it. In the card game, this corresponds to always picking at random a column among those with fewest cards and then pointing at a random card in that column. The other algorithm considered, SC (SHORT CLAUSE), is the same as GUC unless only 3-clauses are present. In that case, rather than picking a random literal from a random clause of length 3, the algorithm satisfies a random literal, i.e. it randomly picks an unset variable and assigns it a random value.

Let $\phi = -2/3\mathbf{W}_{-1}(-e^{-3}) = 3.003...$ be the unique solution of $3r/2 - \ln r = 3 + \ln(3/2)$ greater than 2/3. The performance of the two algorithms on random 3-SAT formulae is very similar: both algorithms succeed with positive probability if $r < \phi$ and both fail w.h.p. if $r > \phi$. The reason for this similarity is that in both algorithms for $r \in (2/3, \phi)$, w.h.p. $\mathcal{S}_2(t) \neq \emptyset$ for all $t \in [n^{1/2}, \alpha n]$, where $\alpha = \alpha(r)$ is such that by time $t = \alpha n$ the remaining formula is "very easy". In the following, for concreteness, we will focus on SC; yet our analysis does not really distinguish between the two algorithms and can be applied to GUC as well.

In order to analyze SC, we first observe that the evolution of 3-clauses is exactly the same as for UC and UCWM since, again, the choice of which variable to set in each step is completely independent of the remaining 3-clauses. To analyze the evolution of 2-clauses let us first focus on the most relevant case $\mathcal{S}_2(t) \neq \emptyset$. In that case, if $\mathcal{S}_1(t) = \emptyset$ the algorithm picks a random 2-clause c and a random literal $\ell \in c$. Note now that in the context of the card game we have the following: after the algorithm has pointed to the card containing the literal ℓ to be satisfied, every clause in $\mathcal{S}_2(t)$ (other than the chosen one c) contains the underlying variable with probability $2/(n-t)$. Therefore, we see that when $\mathcal{S}_1(t) = \emptyset$ and $\mathcal{S}_2(t) \neq \emptyset$,

$$\mathbf{E}(\Delta C_2(t) \mid \mathbf{H}(t)) = \frac{3C_3(t)}{2(n-t)} - \frac{2(C_2(t)-1)}{n-t} - 1 = \frac{3C_3(t)}{2(n-t)} - \frac{2C_2(t)}{n-t} - 1 + o(1) \ , \qquad (6)$$

where the second equality assumes that $n - t = \Omega(n)$. Thus, we see that SC takes a more "conservative" approach than UCWM: it trades the potential benefit of picking the smaller of the two possible $(3 \to 2)$-flows with the certainty of removing an "extra" 2-clause.

To remove the conditioning on $\mathcal{S}_1(t) = \emptyset$, similarly to the case for UCWM, we will apply the "lazy-server" idea so that rather than always satisfying some unit clause when unit clauses exists, the algorithm instead only attempts to satisfy unit clauses "at the appropriate rate". Thus, the only issue that remains to be addressed is determining the conditions under which $\mathcal{S}_2(t) \neq \emptyset$.

As we mentioned above, in a run of SC on $F_3(n, rn)$ for $r \in (2/3, \phi)$ we have $\mathcal{S}_2(t) \neq \emptyset$ except, perhaps, when $t = o(n)$ or $t > \alpha n$. In fact, something stronger is true: for every fixed $\epsilon > 0$, there exist b_ϵ, e_ϵ such that w.h.p. for all $t \in [b_\epsilon n, e_\epsilon n]$, $C_2(t) > \epsilon n$; rather naturally, $\lim_{\epsilon \to 0} b_\epsilon = 0$. The reason for this is that in the beginning of the execution, and for quite a while, the expected change

[2] We solved this differential equation with interval arithmetic, getting *provably* correct results.

[3] The improvement over the analysis in [12] is due to the fact that we are using a sharper bound for $M(\lambda)$.

in $C_2(t)$ is strictly positive, e.g. initially it is at least $3r/2-1$. As a result, $C_2(t)$ behaves like a random walk with positive drift (and a reflecting barrier). In the beginning, it might "return to the origin" ($C_2(t)=0$) a few times, but within $o(n)$ steps it "takes off" and does not return until much later, well after the drift has become negative.

Recall now that Theorem 8 requires that we specify a domain $D \subset \mathbb{R}^{k+1}$ (for the normalized parameters of the process) inside which the process is "well-behaved". For example, in our earlier applications of the theorem, the only requirement for the process to be well-behaved was that $t \le (1-\epsilon)n$ for some constant $\epsilon > 0$, mandated by the need for a Lipschitz condition on the functions describing the expected change in each step (the derivatives). The theorem then asserts that if we start the process at any configuration inside the domain, the solutions of the differential equations with the corresponding initial conditions give a good approximation of the (normalized) process (for at least) as long as these solutions remain inside the domain. Here, good behavior will also require that $\mathcal{S}_2(t) \neq \emptyset$ since this is the typical/interesting case and it allows us to get a handle on the dynamics of the 2-clauses. Since good behaviour can only be cast in terms of normalized quantities, the additional requirement will be that $C_2(t) > \delta n$, for some arbitrarily small $\delta > 0$. Consequently, to start the process inside the domain we will add $2\delta n$ random 2-clauses to the input formula. These will provide a "safety cushion" which will guarantee that w.h.p. the process does not run out of 2-clauses "prematurely". Moreover, since δ can be chosen arbitrarily small, these clauses only add an inconsequential burden to the algorithm.

It is also worth pointing out that to setup the domain we do not need to predict/control when the process will run out of 2-clauses. It suffices for us to only specify the properties defining the domain and make sure that the process initially satisfies them; we can read off the point where each trajectory leaves the domain from the solution of the corresponding differential equation.

Bringing these considerations together with (6) and the lazy-server idea, all in all, we get

$$\begin{aligned}
\mathbf{E}(\Delta C_2(t) \mid \mathbf{H}(t)) &= \frac{3C_3(t)}{2(n-t)} - \frac{2C_2(t)}{n-t} - 1 \times \left(1 - \min\left\{1, (1+\theta)\frac{C_2(t)}{n-t}\right\}\right) + o(1) \\
C_2(0) &= \lfloor 2\delta n \rfloor \ .
\end{aligned}$$

Recalling that $C_3(t) = c_3(t/n)\cdot n + o(n)$, where $c_3(x) = (1-x)^3$, yields

$$\frac{dc_2}{dx} = \frac{3}{2}r(1-x)^2 - (1-\theta)\frac{c_2(x)}{1-x} - 1 \tag{7}$$

$$c_2(0) = 2\delta \ . \tag{8}$$

To conclude the analysis we will solve the above differential equation and determine r, x_e such that i) for all $x \in [0, x_e]$: $c_2(x) > \delta$, and $c_2(x)/(1-x) < 1$, and ii) $c_2(x_e) + c_3(x_e) < (1-x_e)$. To do this it will suffice to consider $\theta = \delta = 0$ and appeal to a standard continuity argument. Letting c_2^* denote the specialization of c_2 when $\theta = \delta = 0$ we have

$$\begin{aligned}
\frac{dc_2^*}{dx} &= \frac{3}{2}r(1-x)^2 - \frac{c_2^*(x)}{1-x} - 1 \\
c_2^*(0) &= 0
\end{aligned}$$

which yields

$$c_2^*(x) = \frac{1}{4}(6rx - 3rx^2 + 4\ln(1-x))(1-x) \ .$$

For each $r > 2/3$, there exists $x_e > 0$ such that $c_2(x) > 0$ for $x \in (0, x_e)$. By considering the derivative of $c_2^*(x)/(1-x)$ we get that for $x \in [0,1]$, $c_2^*(x)/(1-x) \le \zeta(r) = 3r/4 - 1/2(1+\ln(3/2))$.

For $r > 2/3$, solving $\zeta(r) = 1$ gives $r = \phi = -2/3\mathbf{W}_{-1}(-e^{-3}) = 3.003...$ Moreover, for all $r \in (3, \phi)$ we have: i) if $x \in (0, 0.89]$ then $c_2(x) > 0$, and ii) if $x_e = 0.85$ then $c_2^*(x_e) + c_3(x_e) < 3/4(1 - x_e)$. Finally, it's not hard to show that for every $\xi > 0$, we can have $|c_2(x) - c_2^*(x)| < \xi$ for all $x \in [0, 1]$ by taking δ, θ sufficiently small. Thus, an argument similar to the one used in previous sections implies $r_3 \geq \phi$.

7 Should we care for the present or the future?

As we discussed in Section 3, UC fails w.h.p. if the rate at which unit clauses are generated ever becomes greater than 1. Thus, it seems reasonable to consider reducing the $(2 \to 1)$-flow as follows: when no 1-clauses exist, pick a variable at random and set it so as to minimize the number of 2-clauses that become 1-clauses. This is the same as UCWM except that now majority is considered among 2-clauses rather than among 3-clauses. Perhaps somewhat surprisingly, this extension of UC yields no improvement at all: it fails w.h.p. for $r > 8/3$, just like UC.

The reason for which biasing the $(2 \to 1)$-flow does not give any improvement is rather simple: the evolution of 2-clauses under this algorithm is identical to their evolution under UC. That is, in every step t (free or forced) if ℓ is the literal chosen to be satisfied, then each clause in $\mathcal{S}_2(t)$ contains one of $\ell, \overline{\ell}$ with probability $2/(n-t)$ and each clause in $\mathcal{S}_3(t)$ contains $\overline{\ell}$ with probability $1/2 \times 3/(n-t)$. Biasing the $(2 \to 1)$-flow does not affect the *total* flow out of bucket 2 (or in it). Thus, if $r > 8/3$ then at $t_b = \lfloor n/2 \rfloor$ the density of 2-clauses exceeds 1 and the algorithm fails w.h.p.

In spite of the above spectacular failure, biasing the $(2 \to 1)$-flow can actually be very useful. In particular, note that in all the algorithms we considered so far, the improvement over UC came from their actions during free steps, since it is in those steps that one can try to minimize the growth of 2-clauses. Now, reducing the $(2 \to 1)$-flow can reduce the number of forced steps the algorithm takes, thus increasing the number of free steps available. In fact, the only problem with our unsuccessful extension of UC above was that while it "generated" many free steps, it did nothing "useful" with them, i.e. nothing that would help reduce the number of 2-clauses.

To take a further step along this path, suppose that in a free step we (somehow) decided to set variable x_5. If it turns out that, say, $\overline{x}_5$ occurs more often than x_5 in both 3-clauses and 2-clauses then it is clearly a good idea to set x_5 to FALSE: this minimizes both the number of 2-clauses generated and the number of 1-clauses generated, the latter yielding (in expectation) fewer forced steps. However, what should we do if, say, x_5 is more frequent in 3-clauses while $\bar{x}_5$ is more frequent in 2-clauses? Setting x_5 to TRUE would then generate fewer 2-clauses while setting it to FALSE would generate fewer 1-clauses. Hence the following tradeoff:

- Minimizing the $(3 \to 2)$-flow is good *now* because this flow determines the number of newly generated 2-clauses and our overall goal is to minimize the number of 2-clauses.
- Minimizing the $(2 \to 1)$-flow is good for the *future* because this flow equals the number of newly generated 1-clauses and the fewer 1-clauses we have, the more opportunity we will have to make good choices in future steps.

7.1 How much is freedom worth?

Motivated by the above discussion, we will consider an algorithm that tries to generate free steps when doing so is not detrimental to the "current" objective of keeping the number of 2-clauses small. This will yield $r_3 \geq 3.165$, in fact giving a small improvement over the "two at a time" algorithm of [1]. The idea is the following. In free steps we will always pick to set a variable v

appearing in a 2-clause c (just like SC), yet we will set v so as to, mainly, minimize the $(3 \to 2)$-flow. That is, we will always set v so as to minimize the $(3 \to 2)$-flow except for when there is a tie in that minimization; in such a case we will set v so as to satisfy c. Note that by picking a variable in a 2-clause *and* minimizing the $(3 \to 2)$-flow we are doing something which is very good "now" since we combine the benefits of SC and UCWM, i.e. we get both the guaranteed removal of an "extra" 2-clause and the minimization of the $(3 \to 2)$-flow. Naturally, this combination comes at a price: whenever we ignore the sign of v in c we bias the $(2 \to 1)$-flow to higher values. This is because, in expectation, the chosen 2-clause becomes a unit clause with probability $1/2$ while every other 2-clause still becomes a unit clause with probability $\frac{1}{2(n-t)}$ (c was guaranteed to be satisfied under SC). Exploiting ties to satisfy c simply attempts to reduce this price.

TIEBREAKER

```
1. For t = 1,...,n

   (a) If there are any 1-clauses
          Pick a 1-clause uniformly at random and satisfy it
   (b) Otherwise, if there are no 2-clauses
         i. Pick an unset variable x uniformly at random
        ii. If x appears positively in at least half the remaining 3-clauses
               Set x = TRUE
       iii. Otherwise
               Set x = FALSE
   (c) Otherwise
         i. Pick a remaining 2-clause c = ℓ1 ∨ ℓ2 uniformly at random
        ii. Pick ℓ ∈ {ℓ1, ℓ2} uniformly at random
       iii. If ℓ appears fewer times than ℓ̄ in the remaining 3-clauses
               Set ℓ = FALSE
        iv. Otherwise, Set ℓ = TRUE
```

It is clear that we could do a bit better in our algorithm if, in case of a tie in the $(3 \to 2)$-flow, we set v so as to minimize the $(2 \to 1)$-flow rather than setting it so as to satisfy c. However, our version is much simpler to analyze and not much worse off. To see this note that ideally we would be selecting the minimum of two random variables $X_1 + 1$ and X_2 where, asymptotically, X_i are i.i.d. Poisson random variables with mean $\lambda = C_2(t)/(n-t) + o(1)$. Since we plan to keep $\lambda < 1$ throughout the algorithm's execution, the strategy of always picking X_2 is not much worse than the optimal one. (For example, for $\lambda < 3/4$ the probability we err is smaller than $1/10$.)

We will present the analysis of TIEBREAKER in an informal, "back of the envelope" style. Relying on experience from previous sections the interested reader might want to verify that our calculations indeed form the skeleton of a complete analysis.

As a first step let us determine the probability that we get a tie in the $(3 \to 2)$-flow. Letting $\frac{3C_3(t)}{2(n-t)} = \lambda$ and recalling that $P(\lambda; i) \equiv \Pr[\mathrm{Po}(\lambda) = i]$, asymptotically, this probability is equal to

$$\mathrm{Tie}(\lambda) \quad = \quad \sum_{i=0}^{\infty} P(\lambda; i)^2$$

$$
\begin{aligned}
&\geq \sum_{i=0}^{s} P(\lambda; i)^2 \\
&\equiv T_s(\lambda) \ ,
\end{aligned}
$$

where the bound given by T_s gets better as s is increased. (The Tie function has a closed form expression only in terms of the modified Bessel function of the first kind.) Similarly to the analysis of UCWM, we will marginally "dumb down" the algorithm to deal with the fact that we only have a bound on the probability of a tie. In particular, when a tie occurs we will have the algorithm toss a coin to decide if it will act on this fact (setting the chosen variable the way it appears in the 2-clause from which it was chosen) or it will assign the variable a random value. By performing the latter step with appropriate probability (that depends on s) the 2-clause dynamics then behave as if the probability of a tie was exactly T_s.

Now, using the lazy-server idea and our bound for Tie we can proceed to determine the rate at which the algorithm should attempt to satisfy unit clauses by "matching" it with the rate at which such clauses will be generated. (We will focus on the most relevant case $\mathcal{S}_2(t) \neq \emptyset$.) Note, of course, that this is a rather cyclical calculation since the rate at which 1-clauses are generated depends on the rate with which the algorithm attempts to satisfy 1-clauses (since this last rate affects $C_2(t)$). Nonetheless, if we arbitrarily set the rate for attempting to satisfy 1-clauses to U, then the expected number of 1-clauses generated per step is

$$
\frac{C_2(t)}{n-t} + (1-U) \times \left(1 - T_s\left(\frac{3C_3(t)}{2(n-t)}\right)\right) \times \frac{1}{2} + o(1) \ .
$$

This is because in every step, every 2-clause (except perhaps for one) becomes a unit clause with probability $1/(n-t)$, giving the $C_2(t)/(n-t) + o(1)$ term; moreover, if the step is free and there is no tie, the chosen clause c becomes a unit clause with probability $1/2$. Now, since U must be "barely greater" than the rate at which 1-clauses are generated, we require that U is $1+\theta$ times the above quantity, yielding

$$
U(t, C_2(t), C_3(t)) = (1+\theta') \times \frac{\frac{2C_2(t)}{n-t} - T_s\left(\frac{3C_3(t)}{2(n-t)}\right) + 1}{3 - T_s\left(\frac{3C_3(t)}{2(n-t)}\right)} \ ,
$$

for some arbitrarily small constant $\theta' > 0$.

We will also need to use a few old tricks. Just like we did for UCWM, we are going to "dumb down" the algorithm in terms of its minimization of the $(3 \to 2)$-flow, so that in free steps the expected value of that flow is *equal* to $B_q\left(\frac{3C_3(t)}{2(n-t)}\right)$. Moreover, just like we did for SC, we are going to add $2\delta n$ random 2-clauses to the original formula to guarantee that w.h.p. the algorithm does not run out of 2-clauses before the residual formula becomes "very easy". Finally, we will use that (just like in every other algorithm) w.h.p. $C_3(t) = (1-t/n)^3 \cdot n + o(n)$. Thus, letting

$$
u(x, c_2(x)) \equiv (1+\theta') \times \frac{\frac{2c_2(x)}{1-x} - T_s\left(\frac{3}{2}r(1-x)^2\right) + 1}{3 - T_s\left(\frac{3}{2}r(1-x)^2\right)}
$$

we get that the differential equation corresponding to $C_2(t)$ under TIEBREAKER is

$$
\begin{aligned}
\frac{dc_2}{dx} &= u(x, c_2(x)) \times \left(\frac{3}{2}r(1-x)^2 - \frac{2c_2(x)}{1-x}\right) + \\
&\quad (1 - u(x, c_2(x))) \times \left(B_q\left(\frac{3}{2}r(1-x)^2\right) - \frac{2c_2(x)}{1-x} - 1\right) \ , \\
c_2(0) &= 2\delta \ .
\end{aligned}
$$

Solving the above differential equation numerically with $\delta = \theta' = 10^{-6}$ and $s = 20, q = 40$ yields that for $r < 3.165$, $c_2(x)/(1-x) < 1 - 10^{-5}$. Moreover, $c_2(x) > (1/2)10^{-6}$ for all $x \in [0, 0.88]$ and finally, $c_2(4/5) + c_3(4/5) < (3/4)(1 - 4/5)$. Thus, arguing similarly to the previous sections, we get $r_3 \geq 3.165$.

8 Discussion

Two questions naturally arise at this point. How far can we take the approach described here? Are there other algorithmic approaches that might succeed?

Very recently, Achlioptas and Sorkin [6] determined the optimal algorithm expressible via the card game among algorithms that set one or two variables in each step. That is, among algorithms where in each round we start with all cards face down, the algorithm picks one or two variables to set, the cards corresponding to these variables are flipped, and finally the algorithm decides how to set the chosen variables based on that information. It turns out that the best thing to do, in terms of variable picking in free steps, is to always point at a random card in a randomly chosen 2-clause (they "always" exist) and, depending on that variable's appearances in 3- and 2-clauses, sometimes also point at the other variable in the chosen 2-clause. With respect to variable setting the algorithm has a continuously changing policy based on determining the optimal tradeoff between the benefit in $(3 \to 2)$-flow and the benefit in $(2 \to 1)$-flow as a function of the current 3- and 2-clause density, i.e. based on the current "price of freedom". The analysis is based on the framework presented here and the constantly changing nature of the algorithm brings to bear the great flexibility afforded by modeling the execution via differential equations.

Given the analysis in [6] it is not clear how much further the card game framework can go. Naturally, one can attempt to consider setting (many) more variables "at a time", but at the price of greatly increased complexity. Moreover, the algorithms already analyzed in the card game seem to hint at its main weakness: the very limited availability of variable-degree information. Note, for example, that card-pointing in free steps is a rather feeble attempt to capture this information: by pointing at a card we are guaranteed that the underlying variable has degree at least one, thus somewhat biasing our choice towards variables of higher degree. This of course is very far from the following rule, known as Johnson's heuristic, that appears to behave quite well on random formulas. Let the weight of a literal ℓ be equal to $\sum_{\{c:\ell \in c\}} 2^{-|c|}$, where $|c|$ denotes the length of clause c. Now, at every free step choose a literal of maximum weight and satisfy it. Clearly, this heuristic falls far outside of the card-game strategy because nearly all the cards must be uncovered to find the weights on any round. Considering random formulas conditional on their degree distribution, by analogy to the case for random graphs [7, 9, 29], might be useful here.

A related algorithmic question is the following. Using the non-rigorous replica method of statistical mechanics, Monasson et al. [30, 31] made the following rather surprising prediction: for any $\epsilon > 0$ and $\theta < 0.71$, a random formula on n variables with $(1-\epsilon)n$ random 2-clauses and θn random 3-clauses is satisfiable w.h.p. Achlioptas et al. [3] proved this assertion for $\theta \leq 2/3$. Using the framework presented here, it can be shown that 2/3 is tight for all algorithms expressible via the card game. For example, both for UC and SC the reader can readily verify that when the 2-clause density equals 1, the 3-clause density equals 2/3 (this is true for all algorithms considered but we only got analytical solutions for UC and SC). In fact, some later analysis using the replica method by Biroli, Monasson and Weigt [8], under certain "natural" assumptions, suggests that 2/3 might be tight. Determining whether 2/3 is indeed tight is an interesting problem in itself and could provide some direction in terms of going beyond the card game representation of algorithms.

Acknowledgements

I want to thank John Franco, Claire Kenyon, Danny Krizanc, Frank McSherry, Michael Molloy, Dana Randall and Ashish Sabharwal for reading earlier drafts of this paper and offering numerous helpful suggestions.

References

[1] D. Achlioptas. Setting two variables at a time yields a new lower bound for random 3-SAT. In *32nd Annual ACM Symposium on Theory of Computing (Portland, OR, 2000)*, pages 28–37. ACM, New York, 2000.

[2] D. Achlioptas, J. H. Kim, M. Krivelevich, and P. Tetali. Two-coloring random hypergraphs. In *RANDOM'00 (Geneva, 2000)*, volume 8 of *Proceedings in Informatics*, pages 85–96. Carleton Scientific, Waterloo, 2000.

[3] D. Achlioptas, L. M. Kirousis, E. Kranakis, and D. Krizanc. Rigorous results for random $(2+p)$-SAT. In *RALCOM '97 (Santorini, 1997)*, pages 1–13, 1997.

[4] D. Achlioptas, L. M. Kirousis, E. Kranakis, D. Krizanc, M. Molloy, and Y. Stamatiou. Random constraint satisfaction: a more accurate picture. *Constraints.* To appear.

[5] D. Achlioptas and M. Molloy. The analysis of a list-coloring algorithm on a random graph. In *38th Annual Symposium on Foundations of Computer Science (Miami, FL, 1997)*, pages 204–212. IEEE Comput. Soc. Press, Los Alamitos, CA, 1997.

[6] D. Achlioptas and G. B. Sorkin. Optimal policies for greedy 3-SAT algorithms. In *41st Annual Symposium on Foundations of Computer Science (Rodondo Beach, CA, 2000).* To appear.

[7] E. A. Bender and E. R. Canfield. The asymptotic number of labeled graphs with given degree sequences. *J. Combinatorial Theory Ser. A*, 24(3):296–307, 1978.

[8] G. Biroli, R. Monasson, and M. Weigt. A variational description of the ground state structure in random satisfiability problems. *Eur. Phys. J. B*, 14:551–568, 2000.

[9] B. Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European J. Combin.*, 1(4):311–316, 1980.

[10] B. Bollobás, C. Borgs, J. Chayes, J. H. Kim, and D. B. Wilson. The scaling window of the 2-SAT transition. 1999. Manuscript.

[11] A. Z. Broder, A. M. Frieze, and E. Upfal. On the satisfiability and maximum satisfiability of random 3-CNF formulas. In *4th Annual ACM-SIAM Symposium on Discrete Algorithms (Austin, TX, 1993)*, pages 322–330. ACM, New York, 1993.

[12] M.-T. Chao and J. Franco. Probabilistic analysis of two heuristics for the 3-satisfiability problem. *SIAM J. Comput.*, 15(4):1106–1118, 1986.

[13] M.-T. Chao and J. Franco. Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the k-satisfiability problem. *Inform. Sci.*, 51(3):289–314, 1990.

[14] V. Chvátal and B. Reed. Mick gets some (the odds are on his side). In *33th Annual Symposium on Foundations of Computer Science (Pittsburgh, PA, 1992)*, pages 620–627. IEEE Comput. Soc. Press, Los Alamitos, CA, 1992.

[15] S. A. Cook. The complexity of theorem-proving procedures. In *3rd Annual ACM Symposium on Theory of Computing (Shaker Heights, OH, 1971)*, pages 151–158. ACM, New York, 1971.

[16] O. Dubois, L. M. Kirousis, and Y. C. Stamatiou. Upper bounds to the unsatisfiability threshold of random 3-SAT formulas: results and techniques. This volume.

[17] P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. pages 609–627. Colloq. Math. Soc. János Bolyai, Vol. 10, 1975.

[18] W. Fernandez de la Vega. On random 2-SAT. 1992. Manuscript.

[19] J. Franco. Results related to threshold phenomena research in satisfiability: Lower bounds. This volume.

[20] J. Franco and M. Paull. Probabilistic analysis of the Davis–Putnam procedure for solving the satisfiability problem. *Discrete Appl. Math.*, 5(1):77–87, 1983.

[21] E. Friedgut. Sharp thresholds of graph properties, and the k-SAT problem. *J. Amer. Math. Soc.*, 12:1017–1054, 1999.

[22] A. M. Frieze and S. Suen. Analysis of two simple heuristics on a random instance of k-SAT. *J. Algorithms*, 20(2):312–355, 1996.

[23] A. Goerdt. A threshold for unsatisfiability. *J. Comput. System Sci.*, 53(3):469–486, 1996.

[24] G. R. Grimmett and D. R. Stirzaker. *Probability and random processes.* The Clarendon Press Oxford University Press, New York, second edition, 1992.

[25] R. Karp and M. Sipser. Maximum matchings in sparse random graphs. In *22nd Annual Symposium on Foundations of Computer Science*, pages 364–375. IEEE Comput. Soc. Press, Los Alamitos, CA, 1981.

[26] L. Kleinrock. *Queueing Systems Vol. I.* John Wiley & Sons, Inc., New York-London-Sydney, 1975.

[27] T. G. Kurtz. Solutions of ordinary differential equations as limits of pure jump Markov processes. *J. Appl. Probability*, 7:49–58, 1970.

[28] T. G. Kurtz. *Approximation of population processes.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, Pa., 1981.

[29] M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random Structures Algorithms*, 6(2-3):161–179, 1995.

[30] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Phase transition and search cost in the $(2+p)$-SAT problem. In *4th Workshop on Physics and Computation, (Boston, MA, 1996).*

[31] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic "phase transitions". *Nature*, 400(6740):133–137, 1999.

[32] N. C. Wormald. Differential equations for random processes and random graphs. *Ann. Appl. Probab.*, 5(4):1217–1235, 1995.

A Appendix

In the statement of Theorem 8, below, asymptotics denoted by o and O, are for $n \to \infty$ but uniform over all other variables. In particular, "uniformly" refers to the convergence implicit in the $o()$ terms. For a random variable X, we say $X = o(f(n))$ *always* if $\max\{x \mid \Pr[X = x] \neq 0\} = o(f(n))$. We say that a function f satisfies a *Lipschitz condition* on $D \subseteq \mathbb{R}^j$ if there exists a constant $L > 0$ such that $|f(u_1, \ldots, u_j) - f(v_1, \ldots, v_j)| \leq L \sum_{i=1}^{j} |u_i - v_i|$, for all $(u_1, \ldots, u_j)$ and $(v_1, \ldots, v_j)$ in D.

Theorem 8 ([32]) *Let $Y_i(t)$ be a sequence of real-valued random variables, $1 \leq i \leq k$ for some fixed k, such that for all i, all t and all n, $|Y_i(t)| \leq Bn$ for some constant B. Let $\mathbf{H}(t)$ be the history of the sequence, i.e. the matrix $\langle \vec{Y}(0), \ldots, \vec{Y}(t) \rangle$, where $\vec{Y}(t) = (Y_1(t), \ldots, Y_k(t))$.*

Let $I = \{(y_1, \ldots, y_k) : \Pr[\vec{Y}(0) = (y_1 n, \ldots, y_k n)] \neq 0$ for some $n\}$. Let D be some bounded connected open set containing the intersection of $\{(s, y_1, \ldots, y_k) : s \geq 0\}$ with a neighborhood of $\{(0, y_1, \ldots, y_k) : (y_1, \ldots, y_k) \in I\}$.[4]

Let $f_i : \mathbb{R}^{k+1} \to \mathbb{R}$, $1 \leq i \leq k$, and suppose that for some $m = m(n)$,

(i) for all i and uniformly over all $t < m$,

$$\mathbf{E}\left(Y_i(t+1) - Y_i(t) | \mathbf{H}(t)\right) = f_i(t/n, Y_0(t)/n, \ldots, Y_k(t)/n) + o(1) \ \ , \ \textit{always};$$

(ii) for all i and uniformly over all $t < m$,

$$\Pr\left[|Y_i(t+1) - Y_i(t)| > n^{1/5} \middle| \mathbf{H}(t)\right] = o(n^{-3}) \ \ , \ \textit{always};$$

(iii) for each i, the function f_i is continuous and satisfies a Lipschitz condition on D.

Then

(a) for $(0, \hat{z}^{(0)}, \ldots, \hat{z}^{(k)}) \in D$ the system of differential equations

$$\frac{dz_i}{ds} = f_i(s, z_0, \ldots, z_k), \qquad 1 \leq i \leq k$$

has a unique solution in D for $z_i : \mathbb{R} \to \mathbb{R}$ passing through $z_i(0) = \hat{z}^{(i)}$, $1 \leq i \leq k$, and which extends to points arbitrarily close to the boundary of D;

(b) almost surely

$$Y_i(t) = z_i(t/n) \cdot n + o(n) \ \ ,$$

uniformly for $0 \leq t \leq \min\{\sigma n, m\}$ and for each i, where $z_i(s)$ is the solution in (a) with $\hat{z}^{(i)} = Y_i(0)/n$, and $\sigma = \sigma(n)$ is the supremum of those s to which the solution can be extended.

Note: The theorem remains valid if the reference to "always" in (i),(ii) is replaced by the restriction to the event $(t/n, Y_0(t)/n, \ldots, Y_k(t)/n) \in D$.

[4]That is, after taking a ball around the set I, we require D to contain the part of the ball in the halfspace corresponding to $s = t/n \geq 0$.

Lemma 9 (Lazy-server Lemma) *Let $F(0), F(1), \ldots$ be a sequence of random variables and denote $f(t) = E(F(t))$. Let $W(0), W(1), \ldots$ be a sequence of independent Bernoulli random variables with density $w(t)$, i.e. $W(t) = 1$ with probability $w(t)$ and 0 otherwise. For a given integer $s > 0$, let $Q(0), Q(1), \ldots$ be the sequence of random variables defined by $Q(0) = 0$ and $Q(t+1) = \max(Q(t) - sW(t), 0) + F(t)$.*

Assume that there exist constants $a, b, c > 0$ such that for any fixed $j \geq i \geq 0$ and any $\delta > 0$,

$$\Pr\left[\sum_{t=i}^{j} F(t) > (1+\delta)\sum_{t=i}^{j} f(t)\right] < \exp\left(-a\delta^b\left(\sum_{t=i}^{j} f(t)\right)^c\right).$$

Then, if for some $\epsilon, \lambda > 0$ and all $t \geq 0$, we have

$$(1-\epsilon)s\,w(t) > f(t) > \lambda,$$

there exist constants C and k depending on $a, b, c, s, \epsilon, \lambda$ such that for every $m \geq 1$,

$$\Pr\left[\max_{0 \leq t < m} Q(t) > \log^k m\right] = O(m^{-2})$$

$$\Pr\left[\sum_{t=0}^{m-1} Q(t) > Cm\right] = O(m^{-2}).$$