On the Algorithmic Satisfiability of the Random k-SAT Problem

A Project Report for the degree of
Honour School of Computer Science Part B

Mikołaj Poliński
Trinity Term 2023

Abstract

We study a variation of the pivotal problem in computer science - the random k-SAT problem. Here, the goal is to find a satisfying assignment of a randomly generated input formula. Interestingly, instances of the random k-SAT problem exhibit a sharp transition, switching from satisfiable to unsatisfiable when α - the ratio of clauses to variables crosses the satisfiability threshold α_k. Despite the lack of rigorous determinations, empirical results have yielded remarkably precise conjectures for the corresponding satisfiability thresholds in random 3-SAT and 4-SAT instances.

One of the approaches for studying the satisfiability threshold is by designing algorithms to find a satisfying assignment since obtaining rigorous guarantees of their success implies the satisfiability of the corresponding k-SAT instance. In this report, we take a closer look at the two most dominant classes of such algorithms for the unresolved cases of random 3-SAT and 4-SAT.

The first class uses simple rules that exploit the formula's local structure to obtain rigorous performance guarantees. The relative simplicity of these rules enables precise analysis, but it also limits their power, which we will show by proving an upper bound on the performance of one of the most-popular such algorithms.

The second class uses elaborate but non-rigorous techniques inspired by statistical physics with great empirical results around the conjectured satisfiability threshold, but their analysis is still an open problem. We present the two most common algorithms from this class and combine them to obtain a polynomial time algorithm with state-of-the-art results.

Finally, we investigate a potential intersection between these two algorithmic classes by designing an algorithm that combines the efficiency and local perspective of one class with the global perspective of the other.

1 Introduction

The k-SAT problem is a fundamental computer science problem, which arises often in practical applications, but also has great theoretical interest since it is one of the most well-studied NP-complete problems. The input to the problem is a Boolean formula ϕ over n variables which is a conjunction of m clauses, where each clause is a disjunction of exactly k distinct literals. The goal is to find an assignment of truth values to variables that satisfies the formula ϕ, i.e., satisfies all of its clauses. The problem generalizes not only other combinatorial problems like graph colouring, but also many problems that arise in a more applied setting, e.g. scheduling and planning problems, software verification, or hardware equivalence checking.

The problem is NP-complete (a classical result by Cook-Levin) and hence believed not to admit a polynomial time algorithm. Interestingly however, there is a surprising mismatch with instances arising in practice where SAT solvers in industry reportedly work well, and can satisfy instances even with tens of millions of clauses and variables [7]. In fact, it was believed that encountering hard instances of k-SAT is extremely unlikely.

To this end, it was a surprising discovery that hard instances for k-SAT can be generated randomly, using the so-called random k-SAT model. In this project, we investigate these instances more closely and consider the state-of-the-art algorithms, the motivation behind them, and then evaluate their performance and propose various refinements. To give more context however about benchmarking and the methods studied, we first explain the current state of the art.

1.1 The random k-SAT model

In this report, we will focus on the so-called random k-SAT model. For integers M, N this is the uniform distribution on the set of k-SAT formulas with M clauses and N variables, denoted by $\Phi_k(N, M)$. To sample a formula ϕ according to $\Phi_k(N, M)$, let $V = \{x_1, \ldots, x_N\}$ denote the set variables of the formula, we choose the M clauses uniformly at random from all $2^k\binom{N}{k}$ combinations. More precisely, pick k distinct variables from V uniformly at random (without replacement), and for each chosen variable x, with probability $1/2$ include x in the generated clause, and otherwise include its negation $\neg x$. The two parameters characterizing an instance of a random k-SAT problem, are N the number of variables, and α the ratio of the number of clauses M to N, so that $M = N\alpha$.

It was observed that for large N the hardness of instances of the random k-SAT is linked to the value of α, that is for large α the random instance is unsatisfiable with high probability, and for small α an instance is satisfiable with high probability. Moreover, the transition from satisfiable to unsatisfiable is believed to be sharp, which lead to the following conjecture:

Conjecture (Satisfiability threshold)**.** *Let $\alpha > 0$ be a constant. For all $k \geq 2$, there exists a satisfiability threshold α_k such that an instance ϕ of the random k-SAT problem satisfies*

$$\Pr[\phi \text{ is satisfiable}] = \begin{cases} 1 - o(1), & \text{if } \alpha < \alpha_k, \\ o(1). & \text{if } \alpha > \alpha_k, \end{cases}$$

where the probability is over the choice of $\phi \sim \Phi_k(N, M)$ with $M = \lfloor \alpha N \rfloor$. In other words, for large N, ϕ is satisfiable if the density $\alpha < \alpha_k$ (with probability close to 1) and unsatisfiable if $\alpha > \alpha_k$ (with probability close to 0).

The conjecture has been proved for $k = 2$ by Chvatal and Reed [5], who showed that $\alpha_2 = 1$. More recently, Ding et al. [6] proved the conjecture for all sufficiently large k[6]. However, these results are heavily based on elaborate probabilistic arguments (based on estimating the number of assignments using sophisticated expectation/variance arguments).

In this project, we focus instead on the algorithmic question: can we find a satisfying assignment of a random formula $\phi \sim \Phi_k(N, M)$ efficiently (in linear or, more generally polynomial time). We are going to focus in particular on the cases $k = 3$ and $k = 4$ which have not been resolved (even probabilistically), and offer room both for theoretical analysis as well as numerical experiments.

1.2 State of the art for $k = 3$ and $k = 4$

Empirical results from the literature suggest that the satisfiability thresholds for $k = 3$ and $k = 4$ satisfy $\alpha_3 \approx 4.267$ [4] and $\alpha_4 \approx 9.93$ [10], respectively. On the theoretical side, there have been various approaches to obtain bounds on α_3 that agree with the aforementioned numerical values. The upper bounds have been established using probabilistic arguments that estimate the number of satisfying assignments. The best known upper bound, by Kirousis et al. [9], states that $\alpha_3 \leq 4.601$. In contrast, the lower bounds have been established algorithmically, i.e. by analyzing the performance of various algorithms on random k-SAT instances and obtaining rigorous guarantees. Currently, the best lower bound on α_3 is 3.52 which was proven by Hajiaghayi and Sorkin [8].

Much less is known from the theoretical side for the satisfiability of random 4-SAT. Many of the approaches mentioned earlier should generalize in principle to $k = 4$, but the

literature is scarce on (rigorous) bounds for α_4. Nevertheless, 4-SAT is still of great interest, as there, the set of assignments satisfying the formula exhibits interesting patterns that are less visible in the case of $k = 3$[10]. From these observations, a new approach to providing algorithmic lower bounds emerged based on powerful message-passing algorithms that exploit these structures. While proving rigorous guarantees of these algorithms remains an open problem, this is still an active area of research. Below, we provide a clearer distinction between these two approaches.

1.3 High-level overview of algorithms for k-SAT

In practice, k-SAT is usually solved using either branching or local-search algorithms. Although their performance is remarkable in applications of interest, for random k-SAT the industrial solvers perform rather poorly empirically and their complexity makes them hard to analyse.

For random k-SAT, there have been two main algorithmic approaches in the literature:

- *Algorithms based on analysing the performance of Local Heuristics.* These algorithms use simple rules that take into account only the local structure around a given variable or clause, e.g., whether a variable appears only positively. This way, it is relatively easy to decide on the assignment of variables. Moreover, the performance of such algorithms can be analysed theoretically and obtain rigorous bounds; for example, the best-known lower bounds on the satisfiability of 3-SAT were proven this way. Naturally, the simplicity of these rules imposes some limitations. We will show it in chapter Theoretical analysis of local heuristics for random k-SAT by showing an upper bound on the performance of one of the most popular such heuristics for 4-SAT - the Unit Clause with Majority. The upper bound we show is substantially below the conjectured value of α_4, therefore showcasing that local heuristics do not perform well close to the threshold.

- *Algorithms based on Global Heuristics.* The more recent algorithmic approach for random k-SAT has emerged from methods in statistical physics. The premise behind these algorithms is set out to achieve the even more formidable task of understanding the whole space of the satisfying assignments of a given formula. Then, the decision of how to set a particular variable v has a more global perspective, e.g., by picking the truth value for v that appears in the majority of the satisfying assignments. Such information is computed via elaborate but highly non-rigorous methods, which are based on iterative message-passing methods. The most well-known such methods are Belief Propagation, and its advanced version, Survey Propagation. These algorithms yield exceptional empirical results [4] and have given the numerical values of α_3 and α_4 stated above, however their rigorous analysis is a major open problem. We will take a look at these methods in chapter State of the art Global Heuristics, where we study in detail Belief and Survey Propagation; the interest in both goes beyond k-SAT since they can be applied to other models as well (e.g. Bayesian networks).

1.4 Our contributions

In this project, we examine the state-of-art algorithms for the random k-SAT problem. We study both classes of algorithms mentioned in Sections 1.3 and propose a way to combine them in order to get the best of both; i.e., the empirical performance of global-heuristic algorithms but at the same time maintaining the features that make local-heuristic algorithms amenable to theoretical analysis.

Precisely, the project is divided into three parts:

- *Theoretical analysis of local heuristics for random k-SAT.* Here, we take a closer look at the Local Heuristics, by presenting two fundamental local heuristics Unit Clause (UC) and Pure Literal (PL), that are a basic ingredient of all heuristics used in practice. We then consider a refinement to Unit Clause - Unit Clause with Majority and prove an upper bound on its performance on random 4-SAT instances, in order to showcase the limitations of this approach.

- *State of the art Global Heuristics.* In this section, we examine Global heuristics and demonstrate the rationale behind the most recent approach in the field. We implement two state-of-the-art algorithms Belief Propagation and Survey Propagation, and confirm their effectiveness in solving random k-SAT instances. Furthermore, we combine these algorithms to produce a novel method that can satisfy instances close to the conjectured threshold. Moreover, this algorithm runs in polynomial time, in contrast to other SP-inspired algorithms which rely on local search techniques like WalkSAT.

- *Belief Propagation on Local Neighbourhood.* In the final section, we attempt to merge the two dominant approaches, by introducing an algorithm inspired by Belief Propagation that combines the benefits of both global and local heuristics. We do that by running message-passing algorithms on local structures. This method presents a potential direction for further research.

1.5 Definitions and Notation

First, let's review some definitions from propositional logic that we will use throughout this report. Recall that a clause is a disjunction of literals, where a literal is either a variable or its negation. If a literal is not negated, we will say it has a positive polarity, and if it is negated, we will say it has a negative polarity. A CNF is a conjunction of clauses. For a CNF formula ϕ, we will write $Vars(\phi)$ for the set of its variables, and for a literal l, we will write $var(l)$ for the underlying variable.

We define an assignment $\mathcal{A}$ to be a mapping from variables to $\{0, 1\}$, and we will write $\mathcal{A}[\![x]\!]$ to evaluate the assignment on variable x. We extend this notation to negation: $\mathcal{A}[\![\neg x]\!] = 1 - \mathcal{A}[\![x]\!]$, and to clauses: for a clause $C = \vee_{i=1}^{k} l_i$ with l_i being its literals, $\mathcal{A}[\![C]\!] = 1$ if $\mathcal{A}[\![l_i]\!] = 1$ for some i, and $\mathcal{A}[\![C]\!] = 0$ otherwise. The natural extension to CNF formula $\phi = \wedge_{i=1}^{M} C_i$ is defined as follows: $\mathcal{A}[\![\phi]\!] = 1$ if for all i, $\mathcal{A}[\![C_i]\!] = 1$, and $\mathcal{A}[\![\phi]\!] = 0$ otherwise. We will say that $\mathcal{A}$ satisfies ϕ if $\mathcal{A}[\![\phi]\!] = 1$; we will call such an assignment a model of ϕ.

When describing the algorithms, we will often use the phrase "satisfy x in ϕ". By that, we mean to set the literal x to 1 in the implicit assignment and accordingly update ϕ: erase each clause containing x since it becomes satisfied, and from each clause containing $\neg x$, erase $\neg x$. This is equivalent to substituting *true* for each occurrence of x in ϕ.

2 Theoretical analysis of local heuristics for random k-SAT

In this chapter, we will introduce the Pure Literal (PL) and Unit Clause (UC) heuristics, which are the most common yet simplest local heuristics. These heuristics alternate between taking *forced* steps and *free* steps, where forced steps are those decisions that cannot result in a wrong assignment, while free steps do not have such guarantees. The forced steps taken by the Pure Literal and Unit Clause are universally safe, which is why

combining these two heuristics is a basic ingredient of much more elaborate heuristics, e.g., Belief Propagation Inspired Decimation, which we will present later. On the other hand, more advanced heuristics utilize free steps more effectively, leading to significant performance improvements. However, the increased complexity introduced by these free steps makes their analysis more challenging

All local heuristics are very efficient, due to the narrow scope of information they are using, and so each decision they make takes constant time. That is why, they are the perfect fit for branching algorithms solving real-world k-SAT instances. However, on their own, they fall short on harder instances of random k-SAT close to the conjectured threshold α_k. To illustrate this phenomenon, we will present a slightly more sophisticated heuristic, Unit Clause with Majority (UCWM), and provide the upper bound on its performance on a random 4-SAT instance. To achieve this, we adapt Achlioptas' UC analysis on k-SAT, as presented in [2], specifically for the case of $k = 4$. Next, we use these results together with some insights from the analysis of UCWM on 3-SAT to prove that UCWM with high probability does not satisfy instances of 4-SAT for $\alpha \geq 5.38$.

2.1 Pure Literal

First, let us define a literal l as *pure* if there is no clause containing its negation, $\neg l$. Note that if a given CNF formula ϕ contains a pure literal l, we can satisfy l in ϕ without changing the satisfiability of ϕ. That is, by satisfying l, we remove all occurrences of the variable $var(l)$ from ϕ, resulting in a formula that is less constrained. The Pure Literal heuristic is based on this observation and is presented in Algorithm 1. Here, the decision taken in free steps is far from optimal, which significantly limits the power of this algorithm. In fact, Broder et al. [3] showed that the Pure literal almost surely fails in finding a satisfying assignment when the ratio of clauses to variables is $\alpha \geq 1.63$.

Algorithm 1: Pure Literal Heuristic

Input: A k-SAT formula ϕ
Output: *True* *True* if found a satisfying assignment, *False* otherwise
while $Vars(\phi) \neq \emptyset$ **do**
  if *exists pure literal* l **then**
    `// Forced Step`
    satisfy l in ϕ;
  else
    `// Free Step`
    $x \leftarrow$ randomly chosen variable in ϕ ;
    Choose uniformly at random $l \in \{x, \neg x\}$ and satisfy it in ϕ;
  end
end
if ϕ *contains empty clause* **then**
  return *False*
else
  return *True*
end

The above algorithm runs in time $O(Nk\alpha)$, as the existence of a pure literal can be checked in $O(1)$ time by maintaining for each variable x the number of clauses containing x and $\neg x$. Moreover, pruning ϕ after satisfying a variable amortizes to $O(k\alpha)$ per iteration.

2.2 Unit Clause

Similarly to PL, UC exploits a very simple observation. Namely, if any clause C contains only one literal l, then any satisfying assignment to ϕ must satisfy l, otherwise, it would contain an empty clause, yielding an unsatisfiable formula. The algorithm for UC is presented in the pseudo-code 3.

Algorithm 2: Unit Clause Heuristic

```
Input: A k-SAT formula φ
Output: True True if found a satisfying assignment, False otherwise
while Vars(φ) ≠ ∅ do
    if φ contains unit clause {l} then
        // Forced Step
        satisfy l in φ;
    else
        // Free Step
        x ← randomly chosen variable in φ ;
        choose uniformly at random l ∈ {x, ¬x} and satisfy it in φ;
    end
end
if φ contains empty clause then
    return False
else
    return True
end
```

UC runs in time $O(Nk\alpha)$, because after satisfying a variable x, the only clauses that require an update are the ones that previously contained x. Therefore, the detection of new unit clauses amortizes to $O(k\alpha)$.

Analysis

Despite the simplicity of the UC heuristic, the analysis of its performance on random k-SAT formulas is far from trivial. Achlioptas [2] presents a very elegant framework for analyzing the performance of a family of heuristics that includes UC. In the following section, we will present the framework by applying it to the UC heuristic on a random 4-SAT formula.

First, let's start with a new notation. Let $C_i(t)$ denote the number of clauses in ϕ that contain i literals after t steps of the UC algorithm, and let $\Delta C_i(t) = C_i(t) - C_i(t-1)$. Similarly, let $V(t)$ denote the set of variables present in ϕ after t steps. Also, throughout this section, we will refer to clauses of size k by k-clauses.

Then, we can present a crucial lemma proved by Achlioptas [2] that is the center of the whole framework:

Lemma (Uniform Randomness)**.** *Let $\phi \sim \Phi_k(N, M)$ be a random k-SAT formula, and consider the formulat ϕ_t after t steps of Unit Clause. Then, the variables remaining in each clause of ϕ_t are uniform over $L(V(t))$, i.e., if a clause has j literals remaining, each one equals $v \in L(V(T))$ with probability $1/|L(V(T))|$ (independently of the others).*

First, let's observe that the algorithm fails only on forced steps, more precisely when ϕ contains clauses $\{l\}$ and $\{\neg l\}$. To ensure that the algorithm does not encounter such a

situation, we will guarantee that the expected number of unit clauses generated on each iteration is less than 1. This way, in expectation, there is at most one unit clause at each step, so a contradiction cannot be derived. Formally, if $\mathbb{E}[\Delta C_1(t)] = C_2(t)/(N-t) < 1-\gamma$, then with high probability, the UC heuristic succeeds.

Analyzing $C_2(t)$ on its own is a hard task. However, Achlioptas [2] proposes a clever idea of modeling $\Delta C_2(t)$ together with the change in clauses of size 3, $\Delta C_3(t)$, and of clauses of size 4, $\Delta C_4(t)$. Here, if we condition on $C_2(t), C_3(t), C_4(t)$, then by Lemma 2.2 we get that:

$$\Delta C_4(t) = -A \tag{1}$$
$$\Delta C_3(t) = B - C \tag{2}$$
$$\Delta C_2(t) = D - E \tag{3}$$

where

$$A \sim Bin(C_4(t), \frac{4}{N-t}),\ B \sim Bin(C_4(t), \frac{2}{N-t}),\ C \sim Bin(C_3(t), \frac{3}{N-t}),$$
$$D \sim Bin(C_3(t), \frac{3}{2(N-t)}),\ E \sim Bin(C_2(t), \frac{2}{N-t})$$

In order to provide some intuition for the equations presented, let's consider the case of $\Delta C_3(t)$. Suppose we satisfy literal l. Then $C_3(t)$ is increased by the number of clauses that previously were of size 4 and contained $\neg l$, and decreased by the number of clauses of size 3 that contained l or $\neg l$. The probability of the former is $\frac{4}{2(N-t)}$, as we require that one of four literals is $\neg l$, and the probability of the latter is $\frac{3}{(N-t)}$, as there are three literals and one of them is l or $\neg l$. Similar reasoning applies to the other cases.

Now, we observe 2 things:

- The probability that $\Delta C_i(t)$ deviates from its expected value is small due to the properties of Binomial distribution.
- $\Delta C_i(t)$ is smooth in t and $C_i(t)$, that is small change in either t or $C_i(t)$ does not change significantly the value of $\Delta C_i(t)$

These observations hint that the values $C_i(t)$ do not diverge significantly from their mean trajectories, which can be rigorously proven using Wormland's Theorem. To do so, we consider $\mathbb{E}[C_i(t)]$ in a continuous setting, by defining $c_i(x) = \mathbb{E}[C_i(Nx)]/N$ for $x \in [0, 1]$. By doing so, $\mathbb{E}[\Delta C_i(Nx)]$ translates to $\frac{d}{dx}c_i(x)$, yielding a system of differential equations describing the evolution of $c_i(x)$. Moreover, here the second observation of smoothness of $\Delta C_i(t)$ can be formalized, i.e., we require that $\frac{d}{dx}c_i(x)$ is k-Lipschitz smooth, for some constant k. Having done that, we can finally apply Wormland theorem, which states that for any $\epsilon > 0$, the solution to the system of differential yields a trajectory of $c_i(x)$, that, with high probability, closely approximates the actual values of a random process for $x \in [0, 1-\epsilon]$.

This way, we obtain following equations for $C_i(t)$ when $t \leq (1-\epsilon)N$:

$$\begin{aligned} C_2(t) &= \frac{3}{2}\alpha(\frac{t}{N})^2(1-\frac{t}{N})^2 + o(N) \\ C_3(t) &= 2\alpha(1-\frac{t}{N})^3\frac{t}{N} + o(N) \\ C_4(t) &= \alpha(1-\frac{t}{N})^4 + o(N) \end{aligned} \tag{4}$$

Recall that our formula is w.h.p satisfiable if $C_2(t)/(N-t) < 1-\gamma$ for all $t \leq N$. The expression $C_2(t)/(N-t)$ is maximized for $t = \frac{2}{3}N$, and this condition implies that UC, within its first $N(1-\epsilon)$ iterations, will not encounter a contradiction if $\alpha < 4.5$.

However, Wormland's theorem does not tell us anything about the behaviour of $C_i(t)$ when $t > (1-\epsilon)N$. To address this, we set $\epsilon = 0.1$ and $t_e = \lfloor (1-\epsilon)N \rfloor$, and find that $C_2(t_e) + C_3(t_e) + C_4(t_e) < \frac{2}{3}(n - t_e)$. Now, we can randomly choose one variable to remove from each clause of size 3 and two variables to remove from each clause of size 4, which yields a 2-CNF formula with a ratio of clauses to variables less than $\frac{2}{3}$. We can then apply the result for 2-SAT[5] to conclude that this new formula is satisfiable with high probability, and so the algorithm succeeds.

On the other hand, if $\alpha > 4.5$, then $C_2(t)/(N-t) > 1+\gamma$, then again by results for 2-SAT we conclude that w.h.p. the formula is unsatisfiable. Thus, the algorithm cannot succeed, which proves the fact UC satisfies random 4-SAT instance if and only if $\alpha < 4.5$.

2.3 Unit Clause with Majority

In this section, we are going to apply Achlioptas framework, to analyze more complicated algorithm Unit Clause with Majority on 4-SAT instances. The algorithm is presented below 3.

Algorithm 3: Unit Clause with Majority

```
Input: A 4-SAT formula φ
while Vars(φ) ≠ ∅ do
    if φ contains unit clause {l} then
        // Forced Step
        satisfy l in φ;
    else
        // Free Step
        x ← randomly chosen variable in φ ;
        if x appears positively in at least half of the clauses of size 4 then
            satisfy x;
        else
            satisfy ¬x;
        end
    end
end
if φ contains empty clause then
    return False
else
    return True
end
```

Here, we present a modification to the UC algorithm, where in the free steps we choose the assignment of x, so that we maximize the number of 4-clauses leaving the formula, and minimize the number of new 3-clauses generated as a result of deleting x. It is a natural improvement to the UC algorithm, where we make our decision more informed. It does improve the performance, as empirical results shown in 2 indicate that UCWM founds satisfying assignments even for random k-SAT formulas with $\alpha = 5.3$. However, the algorithm clearly fails for larger α, which illustrates the limits of local heuristics. More,

formally

Theorem (Upper Bound on UCWM)**.** *For $\alpha > 5.38$,*

$$\Pr[\textit{UCWM satisfies } \phi] = o(1)$$

where the probability is over the choice of $\phi \sim \Phi_4(N, M)$ with $M = \lfloor \alpha N \rfloor$. That is, Unit Clause with Majority fails with positive probability on random 4-SAT instances with $\alpha > 5.38$.

Below, we present a proof sketch that builds on the results of Achlioptas of UC on 4-SAT and draw inspiration from the results of UCWM on 3-SAT. We omit some minor technical details in order to provide a clearer analysis.

Proof. Notice that optimization introduced to the free steps does not affect the equations for $\Delta C_2(t)$ and $\Delta C_4(t)$. The trajectory of $\Delta C_4(t)$ remains unchanged since we choose x randomly, and the number of clauses leaving C_4 is still the sum of positive and negative occurrences of x. Therefore, equality 1 still holds. On the other hand, since the choice of the polarity of x is independent of the state of clauses of size 2, $\Delta C_2(t)$ is still described by equation 3.

However, in the case of $\Delta C_3(t)$, equation 2 no longer applies because the number of new clauses of size 3 is dependent on whether we are in the forced step or the free step. In the former case, it is still distributed according to $Bin(C_4(t), \frac{2}{n-t})$, but in the latter, it is distributed according to $\min\{Bin(C_4(t), \frac{2}{n-t}), Bin(C_4(t), \frac{2}{N-t})\}$, since the less frequent polarity is removed from 4-clauses. As a result, we encounter two problems: it is difficult to determine if we are in the free or forced step, and the distribution of new 3-clauses does not scale to the continuous setting because it depends on N - the number of variables.

We will address the first issue by following the "lazy-server" idea proposed in [1]. Instead of satisfying a unit clause as soon as it arrives, we will toss a biased coin with probability u of heads. If it is heads, we attempt to satisfy a unit clause; if it does not exist, we satisfy a random literal; otherwise, we perform a free step. This way, we can accurately track whether we are in a free or forced step, as it depends solely on some Bernoulli distribution.

To ensure success, we can apply the Lazy Server Lemma proven in [1]. It states that if messages arrive in the queue with rate λ from some distribution with non-extreme variance, and the server handles on average w messages per step, and $w > \lambda$, then the queue remains bounded. In our case, the messages are unit clauses, and the event of handling a message corresponds to the algorithm satisfying a unit clause. Observe that the probability of not generating an empty clause in a single step with $a > 0$ unit clauses is equal to

$$\left(1 - \frac{1}{2(N-t)}\right)^{a-1}. \tag{5}$$

Therefore, if the number of unit clauses is bounded, i.e., $\sum_{t'=0}^{t} C_1(t') < Mt$ for some constant M, then the probability of not generating an empty clause in the first $0.9N$ steps is at least

$$\left(1 - \frac{1}{0.2N}\right)^{M0.9N} = e^{-4.5M}, \tag{6}$$

that is, it is bounded away from 0.

To guarantee that the number of unit clauses is bounded, we need $w > \lambda$; in our case, we have $w = u$ and $\lambda = C_2(t)/(N-t)$. Hence, if we require $C_2(t)/(N-t) < 1$ just as in

the UC, and set $u = \min\{1, (1+\theta)C_2(t)/(n-t)\}$, we deduce that the algorithm succeeds with positive probability.

Having addressed the issue of forced steps and free steps, we deal with the distribution of new 3-clauses. First, recall that we are considering the formulas with $n \to \infty$. Hence,

$$\min\{Bin(C_4(t), \frac{2}{N-t}), Bin(C_4(t), \frac{2}{N-t})\} \to \min\{X, Y\}, \tag{7}$$

where $X, Y \sim Pois(2C_4(t)/(N-t))$. Here, we got rid of the dependence of n, but another problem arises, as the expectation of $\min\{X, Y\}$ does not admit a closed formula. In [2], Achlioptas upper bounds this expectation and continues the analysis assuming that the algorithm generates more 3-clauses than in reality. His upper bound no longer holds for 4-SAT; hence we note the following

$$B_q = \sum_{x=0}^{q}\sum_{y=0}^{q} \mathbb{P}(X=x)\mathbb{P}(Y=y)\min\{x,y\} \leq \mathbb{E}[\min\{X,Y\}], \tag{8}$$

which can be derived from the definition of expectation. This way, we obtain, in a sense, an optimistic analysis of UCWM, as we underestimate the number of new 3-clauses. Therefore, $\mathbb{E}[\Delta C_4(t)]$ is described by a following relation:

$$\mathbb{E}[\Delta C_4(t)] = u \cdot \frac{2C_4(t)}{N-t} + (1-u)\cdot B_q \tag{9}$$

Again, we translate the relations into a continuous setting and then plug them into the Numerical Differential Equation solver. On plot 1, we present the the behaviour of $C_2(t)/(N-t)$ for $\alpha = 5.38$, $q = 40$ and $\theta = 10^{-6}$. The numerical analysis shows that $C_2(0.675593N)/(N-t) > 1$. Thus, an empty clause is generated over the course of algorithm runtime, i.e. the formula is unsatisfiable with high probability. Therefore, the algorithm Unit Clause with Majority does not succeed for $\alpha > 5.38$, which agrees with the empirical performance of Unit Clause with Majority on random instances presented in the figure 2. □

3 State of the art Global Heuristics

In the previous section, we demonstrated local heuristics that can be very efficiently implemented, and one can rigorously analyze them. However, these benefits come at a cost of a significantly weaker performance on larger ratios of clauses to variables. In this section, we will present global heuristics, which are algorithms that in their free steps, make decisions motivated by a global structure of the formula rather than a local neighborhood. We will focus on the most popular heuristics: Belief Propagation (BP) and Survey Propagation (SP), both exhibiting remarkable performance on large formulas with a ratio close to the conjectured threshold of $\alpha_3 = 4.267$ [4] and $\alpha_4 \approx 9.93$[10].

More precisely, these algorithms aim at computing marginals of a uniform distribution over the set of all satisfying assignments of given k-CNF ϕ. The distribution μ is defined as follows:

$$\mu(\mathcal{A}) = \frac{1}{Z}\mathcal{A}[\![\phi]\!] \tag{10}$$

where Z is the total number of satisfying assignments. This way, the marginal $\mu(x = b)$ for $b = 0, 1$ represents the probability that arbitrary assignment with $x = b$ satisfies ϕ.

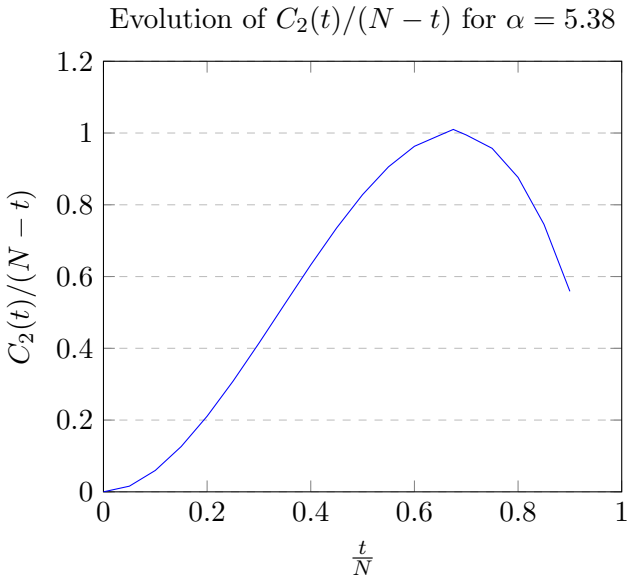

Figure 1: The evolution of $C_2(t)/(N-t)$ for $\alpha = 5.38$. The plot approaches 1 at around $t/N = 0.65$, and the numerical analysis showed that it is maximized for $\frac{t}{n} = 0.675593$, and then $C_2(t)/(N-t) > 1$. Therefore, the algorithm fails with positive probability.

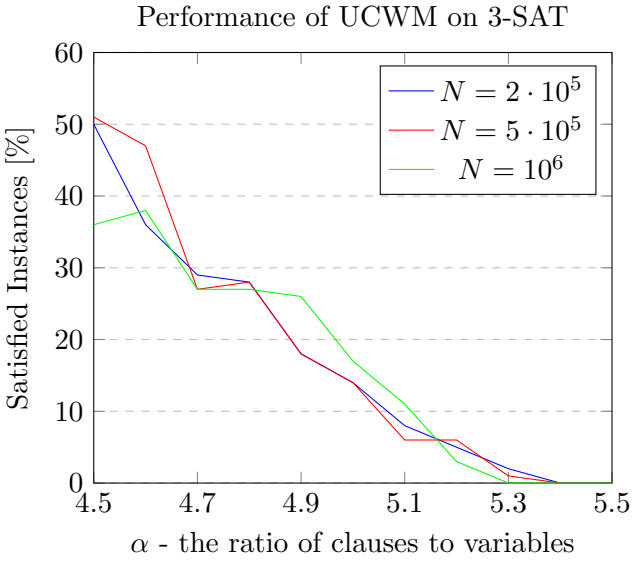

Figure 2: The fraction of 100 random 4-SAT instances that were satisfied by UCWM on 4-SAT with $N = 2 \cdot 10^5$ and $\alpha = 4.5, \ldots, 5.5$. We see that UCWM no longer succeeds on instances with $\alpha > 5.35$, which agrees with the theoretical bound we established saying that UCWM fails on instances with $\alpha > 5.38$. Moreover, we see that bound is rather tight.

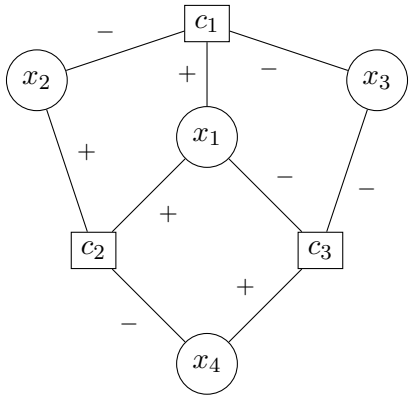

Figure 3: An example Factor Graph for a formula $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$.

Given such marginals, an idea for a heuristic follows immediately. Namely, in each iteration we choose variable x, and set x to 1 if $\mu(x = 1) \geq \mu(x = 0)$, and to 0 otherwise. Both BP and SP exploit that idea but approach the computation of marginals in a different fashion. Before jumping into algorithms, let us first introduce some new concepts.

3.1 Factor Graph

Here, we introduce a popular graphical model of CNF formulas that is useful for message passing algorithms. We define a factor graph of a CNF formula as an undirected bipartite graph with variables and clauses as its vertices, and edges between variables and clauses if a given variable appears in a given clause. Additionally, we mark all edges between any variable x and any clause c with either a $+$ or $-$ sign, depending on whether x appears positively or negatively in c. Please refer to figure 3 for a example.

For vertex v, we define ∂v as the set of all vertices adjacent to v. If v is a variable, then ∂v is the set of all clauses that contain v, and if v is a clause, then ∂v is the set of all variables inside v. Furthermore, we define $\partial_+ v$ (respectively $\partial_- v$) as the set of all vertices connected to v with an edge with label $+$ (resp. $-$). Moreover, for two vertices v and u connected by an edge, we write $\partial_+ v(u)$ (resp. $\partial_- v(u)$) to be the set of all vertices connected to v with an edge with label that agrees (resp. disagrees) with the label of (v, u). We will use these notation usually with v being a variable and u a clause, then $\partial_+ v(u)$ is the set of all clauses that contain v with the same polarity as in u.

3.2 Dynamic Phase Transition

We will also briefly present the other phase transition that k-SAT exhibits, called the Dynamic Phase Transition. Later, it will come in handy while analyzing the empirical results.

Before we do that, define a Hamming distance with respect to two assignments to be $d_\phi(\mathcal{A}, \mathcal{A}') = \sum_{x \in Var(\phi)} |\mathcal{A}[\![x]\!] - \mathcal{A}'[\![x]\!]|$, moreover, we say that two models $\mathcal{A}$ and $\mathcal{A}'$ are *close* if $d_\phi(\mathcal{A}, \mathcal{A}') << N$. So then, by *cluster* we mean an equivalence class in the transitive closure of *close* relation.[11]. That is, a cluster is a connected component in the graph defined over assignments, where there is an edge connecting two assignments if they differ at a small subset of the variables.

According to many statistical physics papers, k-SAT undergoes another transition at threshold α_d^k ($\alpha_d^3 = 3.86$, $\alpha_d^4 = 9.38$[10]). It is conjectured and supported by numerical

experiments that below this threshold, most of the solutions belong to a single large cluster. In [10], this threshold was formally defined as the smallest ratio of clauses to variables α for which a single variable becomes correlated with variables far away in the factor graph. In [10], this condition was named an *extremality* condition, and we will adopt this terminology throughout this report. It is widely, believed that *extremality* condition guarantees a success of BP.

More interestingly, above α_d^k, the solutions form an exponential number of evenly sized clusters. While the extremality condition no longer globally holds, it was shown that within each cluster, this condition still holds. This observation is exploited by SP, which we will cover later.

3.3 Belief Propagation

BP aims at computing marginals of this distribution, namely $\mu(x = b)$ for $b = 0, 1$. Note that such probability can be easily computed if the factor graph is tree. It could be done by a recursive procedure that computes μ for the subtrees which itself correspond to a smaller CNF formula. BP is motivated by this idea, but focuses on tracking violations inside the formula. More precisely, it defines two values:

- $h_{i\to a}$: the probability that variable i violates clause a in the case of absence of a. Note that is proportional to the probability that all clauses b containing i with the same polarity as a do not require i to satisfy them.
- $u_{a\to i}$: the probability that all variables in a except for i are in a state that violates a. That quantity can be easily calculated as the product of all $h_{j\to a}$ for $j \neq i$.

These values are often referred to as *messages*, as they represent the information flowing from clauses to variables and vice versa. Above messages are related through the following equations:

$$h_{i\to a} = \frac{\prod_{b\in\partial_+ i(a)}(1-u_{b\to i})}{\prod_{b\in\partial_+ i(a)}(1-u_{b\to i}) + \prod_{b\in\partial_- i(a)}(1-u_{b\to i})} \tag{11}$$

$$u_{a\to i} = \prod_{j\in\partial a\setminus\{i\}} h_{j\to a} \tag{12}$$

Having computed $h_{i\to a}$ and $u_{a\to i}$, one can calculate *biases*.

$$\mu_x(1) = \frac{\prod_{b\in\partial_- i}(1-u_{b\to i})}{\prod_{b\in\partial_+ i}(1-u_{b\to i}) + \prod_{b\in\partial_- i}(1-u_{b\to i})} \tag{13}$$

$$\mu_x(0) = \frac{\prod_{b\in\partial_+ i}(1-u_{b\to i})}{\prod_{b\in\partial_+ i}(1-u_{b\to i}) + \prod_{b\in\partial_- i}(1-u_{b\to i})} \tag{14}$$

If the factor graph is tree, then the biases equal to the exact marginals, i.e. $\mu_x(b) = \mu(x = b)$ for $b = 0, 1$, and can be computed by first propagating the messages bottom-up and then top-down[4]. However, in general, with finite N and large α the factor graph induced by a random k-SAT formula is not a tree. On the other hand, it was shown that as N tends to infinity, the factor graph of such formula is indeed a tree with a very regular structure[12]. That hints that for large, but finite, N the factor graph should have a tree-like shape, and so the marginals arising from above messages should produce a good approximation of the actual marginals.

This intuition is the driving motivation behind BP, which applies 11 and 12 repeatedly onto the same graph, until it converges to a fixed point. In Belief Propagation, we present pseudocode calculating the biases.

Algorithm 4: Belief Propagation

Input: A k-SAT formula ϕ
Output: Biases $\mu_x(b)_{i=1}^n$
Initialize messages $u^0_{a\to i}$ to 0 and $h^0_{a\to i}$ to 1/2 ;
for $r = 1, \ldots, r_{maxiter}$ **do**
  Calculate messages $u^r_{a\to i}$ and $h^r_{i\to a}$ according to 11 and 12;
  Compute the biases $\mu^r_x(b)$;
  if $\max_{x,b} |\mu^r_x(b) - \mu^{r-1}_x(b)| < \epsilon$ **then**
    Break
  end
end
return $\{\mu_x(b)\}_{i=0}^n$

Empirically, it was shown that these biases produce a good estimate of the actual marginals, however rigorous proof remains an open problem. Nevertheless, they provide us with a valuable insight that we can exploit in the design of a heuristic. More precisely, at each step, if neither PC rule nor UC rule applies, we choose the most biased variable with respect to $\mu_x(b)$, and set it accordingly. This idea is captured in the Belief Propagation Inspired Decimation.

Algorithm 5: Belief Propagation Inspired Decimation

Input: A k-SAT formula ϕ
Initialize messages $u^0_{a\to i}$ to 0 and $h^0_{a\to i}$ to 0.5 **for** $t = 1, \ldots, n$ **do**
  if *there exists pure literal* l *or unit clause* $\{l\}$ **then**
    satisfy l;
  else
    Compute $\{\mu_{x_i}(b)\}_{i=1}^n$ by running BP on ϕ;
    Choose the most biased variable x^*, i.e. $x^* = \arg\max_x |\mu_x(0) - \mu_x(1)|$;
    Set x^* to $b \in \{0, 1\}$ that maximizes $\mu_{x^*}(b)$ and adjust ϕ accordingly;
  end
end
return $True$ if ϕ satisfied, $False$ otherwise

In our implementation, we set $r_{maxiter}$ to 200 and ϵ to 10^{-6}, and we also apply following speed-ups:

- We transform equations 11 and 12 into their log-likelihood form, more suitable for floating point arithmetic.
- Instead of initializing $u^0_{a\to i}$ and $h^0_{i\to a}$ to 1/2 in every call to BP, we reuse the final messages from previous propagation, which speed-ups the convergence of BP.
- We run BP only having satisfied $\lceil f * N_t \rceil$ variables, since the previous propagation, where N_t is the number of variables left at step t. This way we reduce the number of calls made to the BP procedure.

Time Complexity

From the previous section about PC and UC, we know that the steps taken by these heuristics run in $O(Nk\alpha)$, so we only need to analyze the complexity of the Belief Propagation algorithm. First, we note that a single round of message updates takes time $O(Nk\alpha)$, as we need to essentially iterate over all clauses. Hence, BP runs in $O(rNk\alpha)$, where r is the number of rounds.

Numerical experiments, as well as theoretical considerations, suggest that $r = O(\log N)$[4], which is motivated by the intuition that the expected distance between two nodes in our graph is $O(\log N)$. Furthermore, due to the fact that I run BP only having satisfied $\lceil f*N_t \rceil$ variables, since the previous propagation, results in $O(\log N)$ calls to BP. Therefore, the overall running time is $O(Nk\alpha \log^2 N)$.

3.4 Survey Propagation

While BP satisfies random 3-SAT instances at $\alpha = 4.10$, its approach is too general to perform well at $\alpha_3 = 4.26$. To push a boundary of satisfiability, one needs a more sophisticated algorithm.

Recall that in the subsection Dynamic Phase Transition, we noted that for $\alpha > \alpha_d^3$, the set of satisfying assignments is partitioned into an exponential number of evenly-sized clusters. In the breakthrough paper [4], Braunstein et al. report that each of these clusters can be summarized by an assignment of variables to one of three states: 0, 1, or $*$. While the interpretation of assigning 0 or 1 is obvious, the state $*$ is often called a "don't care state"[4], and it indicates that a given variable can be assigned either 0 or 1 without significantly impacting our chances of finding a solution.

Survey Propagation utilizes this observation, and so, satisfies instances of random 3-SAT even with $\alpha = 4.24$[4]. It aims at calculating the distribution defined over clusters, instead of assignments. That is, we compute marginals $\mu(x = b)$ for $b = 0, 1, *$, representing a probability that arbitrary assignment from variables to 0, 1, or $*$ represents a cluster of assignments satisfying ϕ.

While presenting the messages for SP, by saying that clause a sends a warning to i, we mean that all variables in a except for i violate a, and so we require i to satisfy the clause a. Then, we have the following messages:

- $\eta_{a\to i}$: a probability of a warning being sent from clause a to the variable i, indicating that all variables except i violate a. Notice that this interpretation is analogous to messages $u_{a\to i}$ of BP.

- $\Pi^u_{i\to a}$: a probability of no warning coming from $b \in \partial_+ i(a)$ and at least one warning coming from $b \in \partial_- i(a)$. That is, there is no warning coming from a clause that requires i to be the same polarity as in a, and at least one warning coming from a clause that requires i to be in the opposite polarity. Hence, i is pushed to a polarity that violates clause a.

- $\Pi^s_{a\to i}$: at least one warning from a clause requiring i in the same polarity as in a, and no warnings from clauses with i in opposite polarity. This can be understood as a probability of i being pushed towards a polarity that satisfies a.

- $\Pi^*_{a\to i}$: probability of a not receiving any warnings at all, i.e., i is leaning towards the $*$ state.

These messages are related through following equations:

$$\begin{aligned}
\eta_{a\to i} &= \prod_{j\in\partial a\setminus\{i\}}\left[\frac{\Pi^u_{j\to a}}{\Pi^u_{j\to a}+\Pi^s_{j\to a}+\Pi^*_{j\to a}}\right]\\
\Pi^u_{i\to a} &= \left[1-\prod_{b\in\partial_- i(a)}(1-\eta_{b\to i})\right]\prod_{b\in\partial_+ i(a)}(1-\eta_{b\to i})\\
\Pi^s_{i\to a} &= \left[1-\prod_{b\in\partial_+ i(a)}(1-\eta_{b\to i})\right]\prod_{b\in\partial_- i(a)}(1-\eta_{b\to i})\\
\Pi^*_{i\to a} &= \prod_{b\in\partial(i)\setminus\{a\}}(1-\eta_{b\to i})
\end{aligned} \tag{15}$$

Note that there is a fourth possible case that has not been mentioned when i is receiving warnings from clauses containing it positively and from clauses that contain it negatively, which implies a contradiction. This way, the probability that variable i violates a, given that there is no contradictions is equal to $\Pi^u_{i\to a}/(\Pi^u_{i\to a}+\Pi^s_{i\to a}+\Pi^*_{i\to a})$, which motivates the equation for $\eta_{a\to i}$.

We can calculate the bias towards each of the states in the following way:

$$\begin{aligned}
W_i^0 &= \frac{\hat\Pi_i^0}{\hat\Pi_i^0+\hat\Pi_i^1+\hat\Pi_i^*}\\
W_i^1 &= \frac{\hat\Pi_i^1}{\hat\Pi_i^0+\hat\Pi_i^1+\hat\Pi_i^*}\\
W_i^* &= \frac{\hat\Pi_i^*}{\hat\Pi_i^0+\hat\Pi_i^1+\hat\Pi_i^*}
\end{aligned}$$

where

$$\begin{aligned}
\hat\Pi_i^0 &= \left[1-\prod_{b\in\partial_- i}(1-\eta_{b\to i})\right]\prod_{b\in\partial_+(a)}(1-\eta_{b\to i})\\
\hat\Pi_i^1 &= \left[1-\prod_{b\in\partial_+ i}(1-\eta_{b\to i})\right]\prod_{b\in\partial_-(a)}(1-\eta_{b\to i})\\
\hat\Pi_i^* &= \prod_{b\in\partial i}(1-\eta_{b\to i})
\end{aligned}$$

The Caveat

Despite its remarkable performance for $\alpha > \alpha_d^k$, for $\alpha < \alpha_d^k$, SP converges to trivial messages with value 0, not giving us any useful information about the solutions. Although the behavior of SP is not yet well understood, this problem is usually attributed to the fact that in that range, there is only one dominant cluster, and so the distribution that SP estimates (which assigns each cluster the same weight independently of its size) tilts towards small clusters that contain only a very small fraction of the solutions.

This is a major caveat, and it does not allow us to immediately construct a decimation procedure, as we did for BP. To overcome this difficulty, SP is often combined with a local search algorithm (e.g. WalkSAT) run after SP starts outputting trivial messages.

Algorithm 6: SP/BP Inspired Decimation

Input: A k-SAT formula ϕ
Output: *True* if ϕ is satisfiable, *False* otherwise
Initialize randomly messages $\eta_{a\to i}$, $\Pi^u_{i\to a}$, $\Pi^s_{i\to a}$, $\Pi^*_{i\to a}$ to values in $[0,1]$;
for $t = 1, \ldots, n$ **do**
  if *there exists pure literal* l *or unit clause* $\{l\}$ **then**
    satisfy l;
  else
    Run SP until convergence;
    if *for all variables* x, $|W^0_x - W^1_x| < \epsilon$ **then**
      return BP Inspired Decimation on ϕ
    end
    Compute biases W^0_x and W^1_x;
    Choose the most biased variable x^*, i.e. $x^* = \arg\max_x |W^0_x - W^1_x|$;
    Set x^* to $b \in \{0,1\}$ that maximizes W^b_x and adjust ϕ accordingly;
  end
  return $True$ if ϕ satisfied, $False$ otherwise
end

However, in this report, we try something novel by combining SP with BP, and we show that it yields very promising results. The algorithm is presented below 6.

The implementation of Survey Propagation is omitted, as it is analogous to Belief Propagation, except that the updates are replaced with equations 15.

Time Complexity

The time complexity of SP/BP Inspired Decimation is $O(N^2 k\alpha \log N)$, i.e. exactly the same as for BP, since the change to the update rules does not introduce any additional overhead. Again, in our implementation, we set $\lceil f * N_t \rceil$ variables simultaneously, yielding time complexity $O(Nk\alpha \log^2 N)$. While the theoretical complexity does not change, actually, Survey Propagation converges much faster, which provides a constant speed up in comparison to BP Inspired Decimation.

3.5 Results

We run BP Inspired Decimation on instances with $N = 10^4, 2 \cdot 10^4$ and $\alpha = 4.1, \ldots, 4.2$ and we plot the results in figure 4. The first thing we observe is that the performance increases with N, and $\alpha = 4.18$ seems to be the threshold of 3-SAT instances satisfiable with BP. By studying the results of SP, we find numerical evidence of the claims stated in [10] saying that BP performs just roughly as SP. We see that BP performs well even above the dynamic phase transition point, indicating that it produces a good estimate of the marginals, even though, the extremality condition no longer holds. Furthermore, looking at the performance on 4-SAT plotted in figure 6, we empirically confirm the hypothesis from [10] that the performance of BP is closely tied to the Condensation Phase Transition - another transition that k-SAT undergoes.

The results plotted on figure 5 show that SP/BP Inspired Decimation satisfies instances even for $\alpha = 4.23$, close to the conjectured threshold of 4.267. While the results are slightly worse than the ones presented in [4], we do not make use of any of the local search algorithms or backtracking procedures, yielding a polynomial running time. Furthermore,

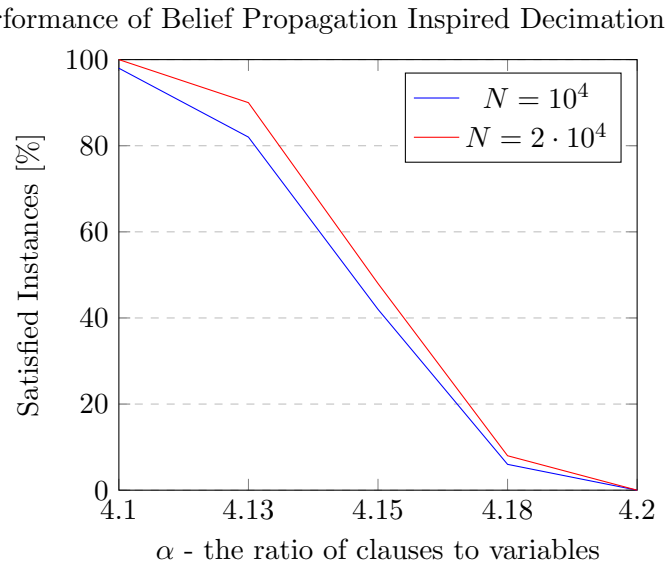

Figure 4: The fraction of instances that were solved after running BP inspired decimation on 50 random 3-SAT instances with $N = 10^4$, $2 \cdot 10^4$ variables and with ratio of clauses of variables $\alpha = 4.1, 4.125, 4.15, 4.175, 4.2$. Here, we see that 4.18 seems to be the largest α for which BP finds solution with positive probability.

on instances not satisfied by the algorithm, the average number of unsatisfied clauses is marginally small; for $\alpha = 4.23$ and $N = 10\ 000$ on the average number of unsatisfied clauses was equal to 2.575 out of 42 300.

The performance of SP/BP might be attributed to the common wisdom claiming that SP converges to the representation of a cluster. More precisely, when the algorithm switches to BP, the state * out-weights the biases towards 0 or 1, i.e., we converged to a cluster. Recall that the extremality condition is believed to hold within these clusters, and so BP is expected to perform well at point. Therefore, after converging to a cluster BP with high probability will find a satisfying assignment.

4 Belief Propagation on Local Neighbourhood

In this section, we present an algorithm that forms a bridge between two previously showcased approaches. The algorithm tries to merge the global perspective with a local one by performing a Belief Propagation in the local neighbourhood. This way, we maintain good performance and potentially obtain an easier analysis. Although we do not attempt the analysis, we present empirical proof that the algorithm performs comparably to other popular heuristics and has the potential to introduce a new rigorous lower bound on α_k.

The Algorithm

The algorithm's primary motivation is that for small d, the local neighborhood in the factor graph forms a tree. This is crucial, as BP computes exact marginals on such graphs. Moreover, these marginals can be computed in just 2 iterations, by first sending messages from the leaves upwards, towards the root, and then sending them from the root downwards, towards the leaves. As we know for large N, the factor graph has a tree-like structure, and so these trees' marginals should be a reasonable estimate of actual

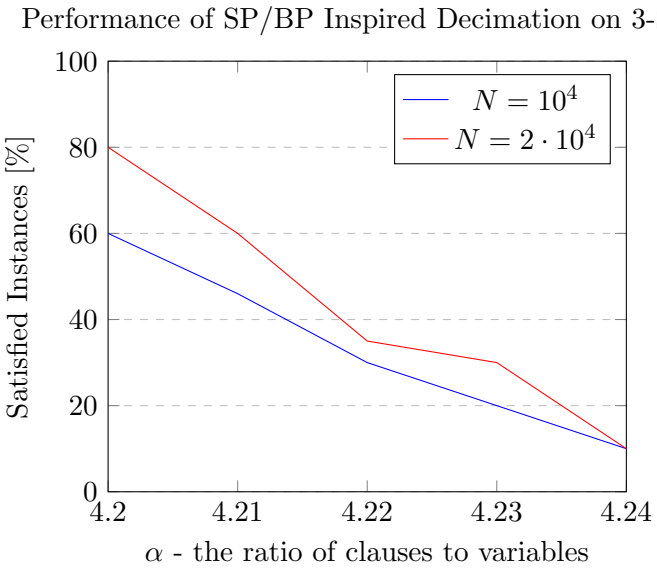

Figure 5: The fraction of instances that were solved after running SP/BP inspired decimation on 20-50 random 3-SAT instances with $N = 10^4$, $2 \cdot 10^4$ variables and with ratio of clauses of variables $\alpha = 4.2$, 4.21, 4.22, 4.23,4.24. Furthermore, on instances not satisfied by the algorithm, the average number of unsatisfied clauses is marginally small; for $\alpha = 4.23$ and $N = 10000$, the average number of unsatisfied clauses was equal to 2.575 out of 423 000.

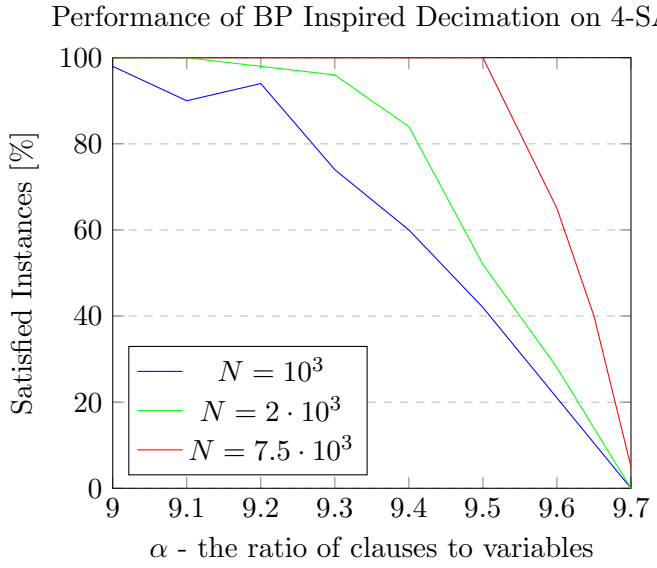

Figure 6: The fraction of instances that were solved after running BP inspired decimation on 50 random 4-SAT instances with $N = 10^3$, $2 \cdot 10^3$ variables and with ratio of clauses of variables $\alpha = 9.0, \ldots, 9.6$. We see that the performance of BP increases with N.

marginals. Still, to further improve the approximation, we run a full Belief Propagation once at the beginning of the algorithm. Then we use obtained values as initial messages for the BP run on the local tree. This way, we first provide the local BP with a global bias, and then, through propagation, it reevaluates the messages according to the local changes. We present the algorithm below, and throughout this section we will refer to by BPLN.

Algorithm 7: Belief Propagation on Local Neighbourhood

Input: A k-SAT formula ϕ, a depth of the neighbourhood $d \in \mathbb{N}$
Run Belief Propagation on ϕ, and save final messages $u^*_{a\to i}$ and $h^*_{i\to a}$;
for $i = 1, 2, \ldots, N$ **do**
  if *there exists pure literal* l *or unit clause* $\{l\}$ **then**
    satisfy l;
  else
    Choose variable x at random;
    Generate formula ψ induced by the neighbourhood of x of depth d in the factor graph of ϕ;
    Run Belief Propagation on ψ to calculate $\mu_x(b)$, using $u^*_{a\to i}$ and $h^*_{i\to a}$ as the initial messages;
    if $\mu_x(0) \geq \mu_x(1)$ **then**
      Set x to 0;
    else
      Set x to 1;
    end
    Prune ϕ;
  end
end
return $True$ if ϕ satisfied, $False$ otherwise

Time Complexity

Again, we can assume that pruning ϕ after fixing a variable takes $O(Nk\alpha)$ time. Moreover, we know that Belief Propagation runs in $O(Nk\alpha \log n)$. Hence, we only need to analyze the complexity of computing the local neighbourhood and running BP on it.

Note that we can compute the formula ψ by running a DFS, which takes time proportional to the size of the neighbourhood. Observe that since ϕ is random, the number of clauses containing x is sampled from $Bin(N\alpha, k/N)$. Therefore, in expectation, the degree of a variable x is αk, and so the number of variables that share a clause with x is, on average $k(k-1)\alpha$. Thus, the expected size of neighbourhood of size d is $O((k(k-1)\alpha)^{d/2})$. This observation, together with the fact that BP requires only 2 iterations to converge, yields an overall running time of $O(Nk\alpha(k(k-1)\alpha)^{d/2}))$. In reality, we run the algorithm with $d = 4$, and the algorithm was significantly faster than the standard BP, due to the large constant hidden in original algorithm.

Results

Define α_{BPT} to be the ratio of clauses to variables, such that for $\alpha < \alpha_{BPT}$ BPLN with high probability satisfies random 3-CNF, and for $\alpha > \alpha_{BPT}$ it does not. We run BP on the local neighbourhood with $d = 4$ on random 3-CNF with ratios $\alpha = 3.0 \ldots, 3.7$, for

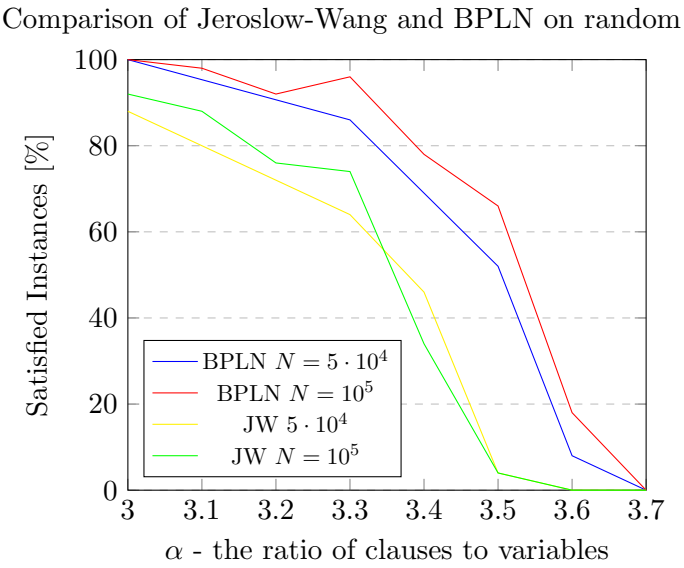

Figure 7: Comparision of Jeroslav-Wang heuristic with Belief Propagation on Local Neighbourhood on 50 random 3-SAT instances with $N = 5 \cdot 10^4, 10^5$ variables and with ratio of clauses of variables $\alpha = 3.0, ..., 3.7$. The lines labeled by BPLN correspond to the performance of BP on Local Neighbourhood, and the label JW corresponds to the Jeroslav-Wang Heuristic. We see that the BP on the local neighbourhood outperforms Jeroslav-Wang heuristic

$N = 5 \cdot 10^4, 10^5$. The results show that for $\alpha = 3.6$, the algorithm satisfies the formula with positive probability, and the probability itself increases with the number of variables. Moreover, the average number of clauses unsatisfied is very low. On the other hand, the BPLN does not satisfy any instances for $\alpha \geq 3.7$, which hints that $\alpha_{BPT} \leq \alpha_d$, and can be intuitively explained by the fact that the above algorithm strongly relies on the fact that far apart variables are uncorrelated, which is guaranteed by the *extremality* condition that holds for $\alpha < \alpha_d$.

We suspect that the analysis could be significantly easier than the analysis of the full BP, as we could use the *extremality* condition to prove strong bounds on the results of the initial BP. Moreover, we think that the changes induced by the local BP are easy to analyze due to the tree structure of the neighbourhood.

Furthermore, the actual running time of the BPLN is much lower than the one of the original BP, as the time complexity of BP hides beneath a large constant induced by the large number of calls to propagation procedure. In fact, during our experiments, on instances with 10 000 clauses, the average execution time of BP on local neighbourhood was ten times faster than the one of BP (on average BP terminated in 25 seconds, while BPLN in 2.3 s). BPLN provides the power and reliability of Belief Propagation, as well as a competitive running time on large instances, thus it might be a good fit as the decision heuristic in branching algorithms.

To test this hypothesis, we compared it to one of the popular decision heuristics used in many branching algorithms - Jeroslav-Wang heuristic, which has been known for for excellent performance on random instances of k-SAT [2]. Figure 7 clearly shows that BPLN outperforms Jeroslav-Wang heuristic. The results are promising, but the idea should be further investigated by comparing BPLN to other decision heuristics, as well as tested in an actual branching algorithm.

Conclusions and Future Work

In this report, we closely examined the algorithmic aspect of the random k-SAT problem. We did that by devoting a chapter to each of the two most promising approaches: local heuristics and global heuristics.

In the section Theoretical analysis of local heuristics for random k-SAT, we studied the foundational approach to solving the random k-SAT problem that uses very simple decision rules. To emphasize the limits of this practice, we introduced the upper-bound on the performance of the well-known algorithm Unit Clause with Majority on random instances of 4-SAT. More precisely, we showed that for random instances with $\alpha \geq 5.38$ UCWM is unsuccessful. There is still a potential for future work here. The expression that we used to bound the expectation of a minimum of two random Poisson variables required us to use numerical solvers. One might find the bound that yields a closed form solution for this system of differential equations, and by doing so, the results might generalize to instances of k-SAT, for $k > 4$. Moreover, one could also slightly adjust the proof to obtain the lower-bound on the performance of UCWM, by finding a lower-bound on the mentioned earlier expected value. By doing so, we would have rigorous guarantees on both the success and failure of UCWM.

In the next section, State of the art Global Heuristics, we presented the motivation behind two revolutionary algorithms Belief Propagation and Survey Propagation. We implemented both methods, and present empirical results that slightly diminish the supremacy of SP which have been built up over the years. On the other hand, we merge both message-passing algorithms to obtain a polynomial algorithm SP/BP that satisfies random 3-SAT instances even at $\alpha = 4.23$ (while $\alpha_3 = 4.26$), which is the first (to our knowledge) realization of SP that does make use of any local search algorithms. The performance should be tested further, as our hardware limitations allowed us to the test the method on instances with only $2 \cdot 10^4$ variables, while breakthrough results where achieved on instances with 10^5 variables.

Finally, in the section Belief Propagation on Local Neighbourhood, we took a look at the intersection of the aforementioned approaches. We propose a polynomial time algorithm BPLN that combines the effectiveness of global heuristics with efficiency of local heuristics. The results, while slightly worse than the ones of SP or BP, beat any other local heuristics, and so it encourages us to test the algorithm as a decision heuristics in industrial SAT solvers. Moreover, we suspect that its rigorous analysis is within the range of modern tools, which might not only tighten the lower bound on α_k, but could also provide valuable insights and directions toward analyzing original BP.

References

[1] Dimitris Achlioptas. Setting 2 variables at a time yields a new lower bound for random 3-sat (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, page 28–37, New York, NY, USA, 2000. Association for Computing Machinery.

[2] Dimitris Achlioptas. Lower bounds for random 3-sat via differential equations. *Theoretical Computer Science*, 265(1):159–185, 2001. Phase Transitions in Combinatorial Problems.

[3] E. Upfal A.Z. Broder, A.M. Frieze. On the satis,ability and maximum satis,ability of random 3-cnf formulas. *Annual ACM-SIAM Symp. on Discrete Algorithms*, 4th, 1993.

[4] A. Braunstein, M. Mezard, and R. Zecchina. Survey propagation: an algorithm for satisfiability. 2002.

[5] V. Chvatal and B. Reed. Mick gets some (the odds are on his side) (satisfiability). In *Proceedings., 33rd Annual Symposium on Foundations of Computer Science*, pages 620–627, 1992.

[6] Jian Ding, Allan Sly, and Nike Sun. Proof of the satisfiability conjecture for large k, 2021.

[7] Vijay Ganesh and Moshe Y. Vardi. *On the Unreasonable Effectiveness of SAT Solvers*, page 547–566. Cambridge University Press, 2021.

[8] MohammadTaghi Hajiaghayi and Gregory B. Sorkin. The satisfiability threshold of random 3-sat is at least 3.52, 2003.

[9] Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, and Yannis C. Stamatiou. Approximating the unsatisfiability threshold of random formulas. *Random structures & algorithms*, 12(3):253–269, 1998.

[10] Florent Krzakała, Andrea Montanari, Federico Ricci-Tersenghi, Guilhem Semerjian, and Lenka Zdeborová. Gibbs states and the set of solutions of random constraint satisfaction problems. *Proceedings of the National Academy of Sciences*, 104(25):10318–10323, jun 2007.

[11] Elitza Maneva, Elchanan Mossel, and Martin Wainwright. A new look at survey propagation and its generalizations. pages 1089–1098, 01 2005.

[12] Andrea Montanari, Federico Ricci-Tersenghi, and Guilhem Semerjian. Solving constraint satisfaction problems through belief propagation-guided decimation. 2007.