Final Project: Review of Schöning's algorithm

Dustin Lin, Shwet Chitnis - CSE200 Fall 2021

December 2021

For our final project we focused on Schöning's k-SAT and CSP algorithm [1]. This paper can be thought as a supplement to help others understand the original paper. We help define terms and concepts Schöning uses in his original proof, and provide more detailed analysis and intermediate steps. We first give an introduction to Schöning's algorithm, as well as define and explain some concepts he uses in his original proof without much explanation. We then go through Schöning's proof of his k-SAT algorithm, as well as the extended version applying to CSPs. Finally, we note a few things about recent developments in SAT algorithms.

1 Introduction

The boolean satisfiability problem (often abbreviated as SAT) is a logic problem of determining if there exists an assignment of variables ($True$ or $False$) that make a boolean formula (built from variables, conjunctions, disjucntions, and negations) true. k-SAT in particular is a special version of SAT that restricts the boolean formula to built from clauses. The main form focused is conjunctive normal form (CNF), a conjunction of disjunction clauses. The k in k-SAT represents how many variables (and negations of them) are in each clause. For example the first equation is a 2-SAT CNF while the second is a 3-SAT CNF.

$$(x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_3) \wedge (\overline{x_1} \vee x_3)$$

$$(x_4 \vee x_1 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_1}) \wedge (\overline{x_1} \vee x_3 \vee x_1)$$

Schöning's k-SAT algorithm is a randomized algorithm that decides if a k-SAT formula is satisfiable for any k. At a high level it guesses an initial assignment bitstring a (where each bit of a represents the assignment of the n variables, with $0 = False$ and $1 = True$. The algorithm repeatedly checks to see if the assignment represented by a satisfies the k-SAT formula, and if not it picks a variable from a unsatisfied clause at random and flips its value.

Algorithm 1 Schöning's k-SAT algorithm [1]

```
procedure SCHÖNING(a formula in k-CNF with n variables)
    Guess an initial assignment a ∈ {0,1}^n          ▷ Uniformly at random
    repeat 3n times
        If the assignment a satisfies the formula, then stop
        Else let C be some clause not satisfied and pick one of the ≤ k variables
        Flip that variables value
```

The constraint satisfaction problem (often abbreviated as CSP) can be thought of as a generalization of SAT. A CSP is a mathematical problem which consists of a set of variables labelled by X, where each variable $X_i \in X$ is defined over a domain $D_i \in D$, and a set of constraints labelled by C. Each constraint $C_i \in C$ is a tuple of variables and some relation defined over the corresponding variables [2]. We say that a CSP has

a satisfying assignment, if for the corresponding values of the variables, Now, any k-SAT instance can be restated as a constraint satisfaction problem. Consider the 3-SAT problem stated above, given by:

$$(x_4 \vee x_1 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_1}) \wedge (\overline{x_1} \vee x_3 \vee x_1)$$

Now, $X = \{x_1, x_2, x_3, x_4\}$, and $D = \{\{0, 1\}\}$ as every variable $x_i \in X$ is defined over $\{0, 1\}$. Consider the following constraints,

- $C_1 = \langle (x_1, x_3, x_4), (x_4 \vee x_1 \vee \overline{x_3}) \rangle$
- $C_2 = \langle (x_1, x_2, x_3), (x_2 \vee x_3 \vee \overline{x_1}) \rangle$
- $C_3 = \langle (x_1, x_3), (\overline{x_1} \vee x_3 \vee x_1)$

The above CSP is only satisfied when each of the constraints is satisfied, i.e, each of the clauses of the 3-SAT instance is satisfied. Therefore, a satisfying assignment for the 3-SAT problem works for the 3-CNF problem stated above and vice versa. Similarly, any k-SAT instance can be converted to a k-CNF, which is a more generalized way of stating satisfying problems. The above algorithm can be modified in order to give a randomized algorithm for any k-CNF problem, as the above stated algorithm does not explicitly use the fact that the individual clauses are disjunctions of the corresponding variables.

1.1 Hamming Distances, Markov chains, and the ballot theorem

Hamming Distance

Hamming distance is a metric widely used in coding theory to measure how far apart two strings over {0, 1} are from each other. The hamming distance between two strings x, y $\in \{0, 1\}^*$ is the the minimum number of bits that must be flipped in order to change one string into the other, i.e, the number of positions in which the two strings differ from each other. For example the 2 bits strings 1001 and 1010 have a hamming of distance of 2, since one would need to flip the third and fourth bit of either string to match the other. The strings here can be used to define many things, like assignments to the variables of a satisfiability problem or a constraint satisfaction problem.

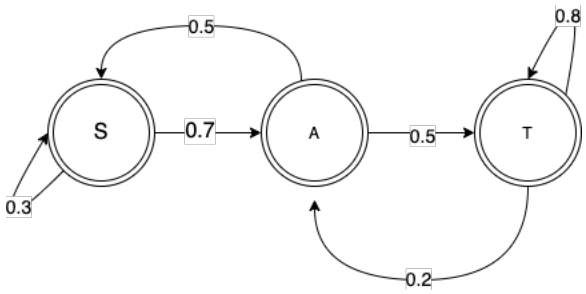

Figure 1: Example of Markov chain diagram

Markov Chains

Markov chains are a model for describing a sequence of probabilistic events. Each event can be thought of as a state, with directed edges connecting to other states each weighted according to the probability that at the current state it will to the next. (Refer to figure 1 for an example) [3].

Bertrand's Ballot Theorem

Bertrand's Ballot Theorem aims at answering the following question: "In an election, candidates A and B receive α and β votes respectively, and A is elected. The total votes between them is $\alpha + \beta$. What is

the probability that during the counting of votes, A always has a lead on B in terms of the number of votes?"

The Ballot Theorem states that the above probability is $\frac{\alpha-\beta}{\alpha+\beta}$. An even more interesting question is how many ways can the ballot be ordered so that A always has a lead on B in terms of the number of votes. Using a variation of the Ballot Theorem, we can see that this probability is $\binom{\alpha+\beta}{\alpha}\frac{\alpha-\beta}{\alpha+\beta}$ [3].

Approximating Binomials

There is a useful fact that Schöning uses later in the proof that we will also state here [4].

$$\binom{n}{\alpha n} \sim 2^{h(\alpha)n} = \left(\frac{1}{\alpha}\right)^{\alpha n}\left(\frac{1}{1-\alpha}\right)^{(1-\alpha)n}$$

Where $h(\alpha)$ is the binary entropy function $h(\alpha) = -\alpha log_2\alpha - (1-\alpha)log_2(1-\alpha)$. The binary entropy function is used due to the problem using bitstrings where each bit takes on a binary value.

2 Schöning's k-SAT algorithm

2.1 Introduction

For his proof, Schöning assumes that a given k-SAT formula has a satisfying assignment (that he calls a^*). There could very well be more than one satisfying assignment for any given boolean formula, but for simplicity he calculates the probability of his algorithm succeeding given that there is only one. If there are multiple satisfying assignments, then this only raises the probability that his algorithm will find such an assignment.

Given what we have learned in class about error reduction of randomized algorithms its important to note that Schöning also mentions that, given a "success probability" p, we will be expected to repeat the algorithm $\frac{1}{p}$ times before finding a satisfying assignment. If we wish to reduce it's error, we merely repeat this algorithm again and again, and the total runtime will remain polynomial in p.

2.2 Definitions of assignments a, a^*, and random variable X

Schöning defines a as the randomly picked initial assignment, as well as a random variable X which counts the number of variables that need to be flipped from a to the satisfying assignment a^*. Since a and a^* are bitstrings of length n (where n is the number of variables in the k-SAT formula) where each bit corresponds to each variable being true or false, X can also be thought of as the *hamming distance* between a and a^*.

To calculate the probability that $X = j$ (ie: that a and a^* differ by j assignments, where $0 \leq j \leq n$), it is important to notice that this value is binomially distributed with parameters n and $\frac{1}{2}$. This is because the "assignemnt" of each bit in a is uniformly and independently random, and there are n assignments which each take on a value of either 1 or 0. Therefore $Pr(X = j) = \binom{n}{j}2^{-n}$. Intuitively, we can think of choosing j bits from a^* (there are $\binom{n}{j}$ different ways of doing so) and flipping their values. The probability that the random assignment a is equal to one of these bitstrings is $\binom{n}{j}$ over all the possible bitstrings 2^n.

2.3 Representing the algorithm as a Markov chain

If we fix $X = j$, then we can represent this process of picking one bit at a time from a flipping its bit value to slowly transform a into a^* as a Markov chain. Schöning sets up states $0, 1, \ldots, n$ each representing the hamming distance between a and a^*, as well as a *start* state. Schöning's algorithm can be represented as this Markov chain where we start at the *start* state, pick an initial assignment a, and move form the start state to one of the states $0, 1, \ldots, n$ based on the hamming distance X. Just like before, the chance that we move from the *start* state to a particular state $0 \leq j \leq n$, is $\binom{n}{j}2^{-n}$. As the algorithm picks bits to flip in a randomly, we move incrementally up and down the the Markov chain depending on if the random

assignment increased or decreased the hamming distance to a^*.

Recall above that after the algorithm picks an initial assignment a, and checks to make sure it is not a satisfying assignment, it will try and pick some clause that is not satisfied. Note that in a k-CNF a clause contains k variables connected with or operators which means that a non satisfying clause has variables whose values are *false*. In our satisfying assignment a^*, we know that this clause is true, so a^* at minimum selects one of these k variables to be true. The probability that the algorithm picks the exact variable to flip to true to match a^* (and thus decrease the hamming distance by 1) is $\frac{1}{k}$, and the probability the algorithm picks another variable to flip (and thus increasing the hamming distance by 1) is $\frac{k-1}{k}$. To relate this back to the Markov chain representation, given that we are at a state j, the probability that we move from state j to $j-1$ and $j+1$ is at least $\frac{1}{k}$, and at most $\frac{k-1}{k}$ respectively.

To calculate the probability that (given we start at an initial state j), the algorithm ends up reaching the "end state" state 0 notice first that this will take at minimum at least j steps. This occurs if the algorithm random picks exactly the j variables to flip to convert a to a^* one after another. If we consider the likely scenario that the algorithm will mess up and choose incorrect variables along the way, Schöning calls this probability q_j. The random walk will make some i steps in the "wrong direction", where $i \leq j$ (increasing the hamming distance), and $i+j$ steps in the correct direction (fixing the i mistakes and making the j changes necessary to stop in state 0). This scenario would result in the algorithm taking $j+2i$ steps (flipping $j+2i$ bits). To calculate this probability that given a starting state j, we will move to state 0 in $j+2i$ steps (making along the way i steps in the "wrong direction"), we can use the ballot theorem.

2.4 Applying the ballot theorem

We can think of candidate A as having $j+i$ votes and candidate B having i votes. We wish to order these votes in such as way that A always has a running total amount of votes larger than B at any given point. In the context of this problem, this translates to ordering our choices of bits to flips such that at any point in our algorithm, we would have moved more often towards state 0 than away from it. We can also think of such an order as prohibiting ourselves from moving *past* state j, since moving to any state greater than j would require B having a running total of votes more than A.

Note here that Schöning is actually undercounting the number of different ways we can move from state j to state 0 using $j+2i$ steps. By using the ballot theorem, we are adding that constraint that the $j+i$ steps in the right direction have to occur more often than the i steps in the wrong direction at any point in the process. Schöning is not taking into account for example, the situation in which the algorithm makes i mistakes first and moves to state $i+j$, and then moving $i+j$ states towards state 0. This wouldn't be counted by the ballot theorem since for the first $i+(i-1)$ "ballots", B would be leading A. This is why Schöning calculates the *lower bound* of the number of different ways one can move from state j to state 0 using exactly $j+2i$ steps.

Using the ballot theorem with $\alpha = j+i$, $\beta = i$, we get the total amount of different orderings is $\binom{j+2i}{j+i} \cdot \frac{j}{j+2i}$.

Note that in his paper Schöning concludes similarly $\binom{j+2i}{i} \cdot \frac{j}{j+2i}$. These two expressions are equivalent based on that fact that for any $k \leq n$, $\binom{n}{k} = \binom{n}{n-k}$. Intuitively this is true if you think about choosing k objects out of n as equal to choosing $n-k$ objects to exclude from n (leaving you with the k you wish to include).

2.5 Lower bounding q_j

The probability q_j that, given that you are at a state j, you move to state 0 in $j+2i$ steps for some $i \leq j$ is lower bounded by:

$$q_j \geq \sum_{i=0}^{j} \binom{j+2i}{i} \cdot \frac{j}{j+2i} \cdot \left(\frac{k-1}{k}\right)^i \cdot \left(\frac{1}{k}\right)^{i+j}$$

Here this expression sums up all the different ways we can order (using the ballot theorem) the $j + 2i$ ways to get from state j to state 0 making $i \leq j$ mistakes. For each $i \leq j$, we calculate using the ballot theorem how many different ways, multiplied by the probability of making i mistakes, and $i + j$ correct choices.

Schöning further lower bounds q_j by the largest term of the sum, when $i = j$:

$$q_j \geq \binom{3j}{j} \cdot \frac{j}{3j} \cdot \left(\frac{k-1}{k}\right)^j \cdot \left(\frac{1}{k}\right)^{2j}$$

He then uses the useful fact of approximating binomials that we outlined above, and makes the claim that the following two expressions are within polynomial factors of one another:

$$\binom{(1+2\alpha)j}{\alpha j} \quad \text{and} \quad \left[\left(\frac{1+2\alpha}{\alpha}\right)^{\alpha} \cdot \left(\frac{1+2\alpha}{1+\alpha}\right)^{1+\alpha}\right]^j$$

We can see the above is true if we set $n = (1 + 2\alpha)j$ and $\alpha_1 = \frac{\alpha}{(1+2\alpha)}$, which allows the use of the approximation outlined by Ash [4]:

$$\begin{aligned}
\binom{(1+2\alpha)j}{\alpha j} \sim 2^{h(\alpha_1)n} &= \left(\frac{1}{\alpha_1}\right)^{\alpha_1 n} \cdot \left(\frac{1}{1-\alpha_1}\right)^{(1-\alpha_1)n} \\
&= \left(\frac{1+2\alpha}{\alpha}\right)^{\alpha j} \cdot \left(\frac{1}{1-\alpha_1}\right)^{(1-\alpha_1)n} \\
&= \left(\frac{1+2\alpha}{\alpha}\right)^{\alpha j} \cdot \left(\frac{1+2\alpha}{1+\alpha}\right)^{\frac{1+\alpha}{1+2\alpha} j}
\end{aligned}$$

*Note that $1 - \alpha = \frac{1+\alpha}{1+2\alpha}$

Schöning then lower bounds q_j yet again, this time replacing the binomial $\binom{3j}{j}$.

$$\binom{3j}{j} \geq \binom{(1+2\alpha)j}{\alpha j} \text{ for } 0 < \alpha \leq 1 \text{ where } \alpha = \frac{1}{k-2}$$

Lower-bounding this way makes intuitive sense, since when $k = 3$, the two binomials are equal (Figure 2). With $k = 3$ this corresponds to $3 - SAT$, where the input formula is restricted to having at most 3 variables per clause. It wouldn't make sense to have $k < 3$, since there are polynomial deterministic algorithms for solving $2 - SAT$.

Thus we can lower bound the largest term of the summation, and using the facts above we can lower bound the binomial $\binom{3j}{j}$ and approximate it to a polynomial degree. Where the last two fractions are also approximated up to a polynomial factor. (α and $\frac{1+\alpha}{2}$ more specifically).

$$\begin{aligned}
q_j &\geq \frac{1}{3}\binom{3j}{j} \cdot \left(\frac{k-1}{k}\right)^j \cdot \left(\frac{1}{k}\right)^{2j} \\
&\geq \left[\left(\frac{1+2\alpha}{\alpha}\right)^{\alpha} \cdot \left(\frac{1+2\alpha}{1+\alpha}\right)^{1+\alpha} \cdot \left(\frac{k-1}{k}\right)^{\alpha} \cdot \left(\frac{1}{k}\right)^{1+\alpha}\right]^j
\end{aligned}$$

We will now show more in detail Schöning's claim that the last inequality is equal to $\left(\frac{1}{k-1}\right)^j$ when $\alpha = \frac{1}{k-2}$.

Note that:

$$\begin{aligned}
\alpha &= \frac{1}{k-2} \\
1 + \alpha &= 1 + \frac{1}{k-2} = \frac{k-1}{k-2} \\
1 + 2\alpha &= 1 + \frac{2}{k-2} = \frac{k}{k-2}
\end{aligned}$$

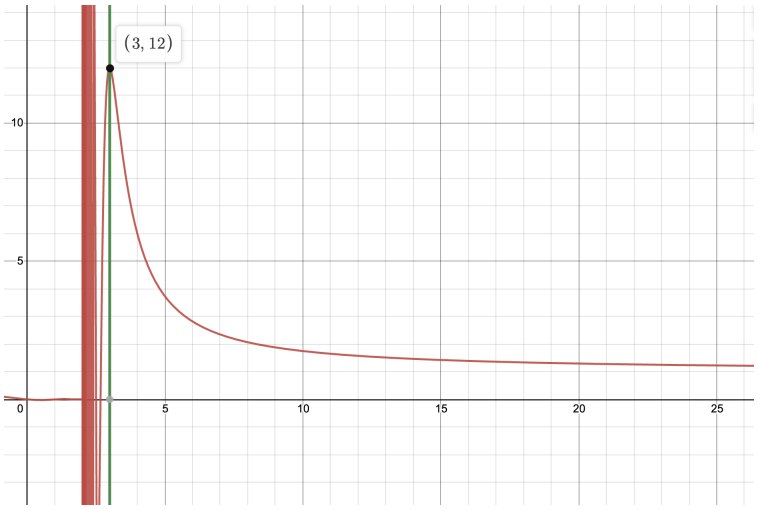

Figure 2: With $j = 2$ the binomial achieves a maximum when $k = 3$

We will only look at the expression being raised to the jth power for simplicity.

$$\left(\frac{1+2\alpha}{\alpha}\right)^{\alpha} \cdot \left(\frac{1+2\alpha}{1+\alpha}\right)^{1+\alpha} \cdot \left(\frac{k-1}{k}\right)^{\alpha} \cdot \left(\frac{1}{k}\right)^{1+\alpha}$$

$$= \left(\frac{(1+2\alpha)^2}{\alpha(1+\alpha)}\right)^{\alpha} \cdot \left(\frac{1+2\alpha}{1+\alpha}\right) \cdot \left(\frac{k-1}{k^2}\right)^{\alpha} \cdot \frac{1}{k} \tag{1a}$$

$$= \frac{(1+2\alpha)^{2\alpha} \cdot (1+2\alpha) \cdot (k-1)^{\alpha}}{\alpha^{\alpha} \cdot (1+\alpha)^{\alpha} \cdot (1+\alpha) \cdot k^{2\alpha} \cdot k} \tag{1b}$$

$$= \frac{\left(\frac{k}{k-2}\right)^{2\alpha} \cdot \left(\frac{k}{k-2}\right) \cdot (k-1)^{\alpha}}{\left(\frac{1}{k-2}\right)^{\alpha} \cdot (1+\alpha)^{\alpha+1} \cdot k^{2\alpha} \cdot k} \tag{1c}$$

$$= \frac{\left(\frac{1}{k-2}\right)^{2\alpha} \cdot k^{2\alpha} \cdot \left(\frac{1}{k-2}\right) \cdot k \cdot (k-1)^{\alpha}}{\left(\frac{1}{k-2}\right)^{\alpha} \cdot (1+\alpha)^{\alpha+1} \cdot k^{2\alpha} \cdot k} \tag{1d}$$

$$= \frac{\left(\frac{1}{k-2}\right)^{\alpha+1} \cdot (k-1)^{\alpha}}{(1+\alpha)^{\alpha+1}} \tag{1e}$$

$$= \frac{\left(\frac{1}{k-2}\right)^{\alpha+1} \cdot (k-1)^{\alpha}}{\left(\frac{k-1}{k-2}\right)^{\alpha+1}} \tag{1f}$$

$$= \frac{\left(\frac{1}{k-2}\right)^{\alpha+1} \cdot (k-1)^{\alpha}}{\left(\frac{1}{k-2}\right)^{\alpha+1} \cdot (k-1)^{\alpha+1}} \tag{1g}$$

$$= \frac{1}{k-1} \tag{1h}$$

(1a) combine terms with equal powers
(1b) expanded to get 1 fraction, distributed exponents
(1c) substituted $(1+\alpha)$ and $(1+2\alpha)$ terms
(1d) expended out the two fractions on the numerator

(1e) canceled out terms
(1f) substituted $(1+\alpha)$ in denominator
(1g) split apart fraction in denominator
(1h) canceled out terms

2.6 Estimating the success probability p

Using q_j, we want to calculate the overall success probability of the k-SAT algorithm. This probability p will take into account all the different hamming distances $0 \leq j \leq n$ as well as the probability that a random assignment a starts in state j. From section 2.2 we know $Pr(X=j) = \binom{n}{j}2^{-n}$, and the probability $q_j = \left(\frac{1}{k-1}\right)^j$. Therefore, the success probability of the k-SAT algorithm is:

$$
\begin{aligned}
p &\geq \sum_{j=0}^{n} \binom{n}{j} \left(\frac{1}{2}\right)^n \left(\frac{1}{k-1}\right)^j \\
&= \left(\frac{1}{2}\right)^n \sum_{j=0}^{n} \binom{n}{j} \left(\frac{1}{k-1}\right)^j \\
&= \left(\frac{1}{2}\right)^n \left[\binom{n}{0} + \binom{n}{1}\left(\frac{1}{k-1}\right)\binom{n}{2}\left(\frac{1}{k-1}\right)^2 + ... + \binom{n}{n}\left(\frac{1}{k-1}\right)^n\right] \\
&= \left(\frac{1}{2}\right)^n \left(1+\frac{1}{k-1}\right)^n \qquad\qquad (1+x)^n = \binom{n}{0} + \binom{n}{1}x + ... + \binom{n}{n}x^n \\
&= \left(\frac{1}{2}\left(1+\frac{1}{k-1}\right)\right)^n \\
&= \left(\frac{1}{2}\left(\frac{k}{k-1}\right)\right)^n
\end{aligned}
$$

The complexity bound of the k-SAT algorithm is $\mathcal{O}\left(\frac{1}{p}\right)$, i.e, $\mathcal{O}\left(\left(2\left(1-\frac{1}{k}\right)\right)^n\right)$
Since we assumed $j + 2i \leq n + 2n = 3n$, the algorithm runs for at most $3n$ steps.

3 Schöning's CSP algorithm

3.1 Notation

Consider a constraint satisfaction problem with the following parameters:

- n - the number of variables
- d - the size of domain
- l - the order of the constraints, which represents the number of variables a constraint depends upon

3.2 Naive Algorithm

A naive algorithm for the above constraint satisfaction problem runs in time $\mathcal{O}(d^n)$. In order to find a satisfying assignment, one can go through all the possible D^n assignments, i.e., all the combinations of values picked from D assigned to $x_1, x_2, ..., x_n \in X$, and then pick an assignment which satisfies all the constraints, if any. A minor improvement in the above time complexity can help solve CSP's much more efficiently, from a practical viewpoint. For example, if the a CSP with the above parameters can be solved in square root time, i.e., in time $\mathcal{O}((\sqrt{d})^n)$, then CSPs with twice as many variables ($2n$) can be solved in the same time complexity ($\mathcal{O}(d^n)$) as stated above.

3.3 Schoning's Algorithm for solving CSPs

Algorithm 2 Schöning's CSP algorithm [1]

1: **procedure** SCHÖNING(a CSP instance)
2: Guess an initial assignment $a \in D^n$ ▷ Uniformly at random
3: **repeat** $3n$ times
4: If the assignment a satisfies the formula, then stop
5: Else let C_i be some constraint not satisfied and pick one of the l variables
6: Assign that variable a random value from the domain D
7:

3.4 Complexity Analysis of Schöning's Algorithm for CSPs

The above algorithm runs in time $\mathcal{O}(d(1-\frac{1}{l})^n)$
Proof of the above complexity:
Consider a CSP with parameters n, l, and d as defined in Section 3.1. First let's consider a result which follows almost immediately from the analysis of Schöning's k-SAT algorithm. If one of the constraints is not satisfied by some assignment $a \in D^n$, there are $l(d-1)$ ways that one of the l variables of the corresponding constraint can be changed. There are l variables involved in the constraint, each of which can acquire one of the $d-1$ values from D (any of the $d-1$ values in D except the variable's current value). Thus, it can be seen that the role of k from the previous analysis has been taken over by $l(d-1)$, as in the k-SAT case, $d=2$ and $l=k$. Therefore, the success probability can be approximated polynomially adequately using the following equation:

$$
\begin{aligned}
p &\geq \sum_{j=0}^{n} \binom{n}{j} \left(\frac{1}{d}\right)^{n-j} \left(\frac{d-1}{d}\right)^{j} q_j && \text{where } q_j \text{ is as computed in the k-SAT analysis} \\
&= \sum_{j=0}^{n} \binom{n}{j} \left(\frac{1}{d}\right)^{n-j} \left(\frac{d-1}{d}\right)^{j} \left(\frac{1}{k-1}\right)^{j} \\
&= d^{-n} \sum_{j=0}^{n} \binom{n}{j} \left(\frac{d-1}{k-1}\right)^{j} \\
&= d^{-n} \left(1+\frac{d-1}{k-1}\right)^{n} && \text{where } k = l(d-1)
\end{aligned}
$$

The above result can be improved for constraint satisfaction problems with $d > 2$. Note: We will not go into the details involved in this analysis, but only give the intuition behind it. Notice that we did not consider the case where the Markov chain moves from a state j to a state j, i.e., it changes the value of one of the wrong variables to a wrong value. The Hamming Distance between the current assignment and the fixed satisfying assignment does not change. Now, consider the Markov chain $j \to j \to j \to j \to ... \to j \to j+1$ (or $j \to j \to j \to j \to ... \to j \to j-1$). Such a Markov Chain can be transformed into the Markov Chain considered above ($j \to j+1$ or $j \to j-1$) by applying some trivial probabilistic transformations. Using such a technique in the analysis, gives an updated formula for the success probability of the algorithm:

$$
\begin{aligned}
p &\geq d^{-n} \left(1+\frac{d-1}{k-1}\right)^{n} && \text{where } k = (l-1)(d-1)+1 \text{ from the updated analysis} \\
&= d^{-n} \left(1+\frac{1}{l-1}\right)^{n} \\
&= d^{-n} \left(\frac{l}{l-1}\right)^{n}
\end{aligned}
$$

which tells us that the complexity bound for Schöning's CSP algorithm is $\left(d\left(1-\frac{1}{l}\right)\right)^n$, which is very similar in form to the complexity bound obtained for Schöning's k-SAT algorithm.

4 Other improved SAT algorithms

Grover's Algorithm

Classically, unstructured search problems require $\mathcal{O}(n)$ time where n is the size of the database in order to find a match with a probability of at least $\frac{1}{2}$. Grover's algorithm exploits the continuous nature of quantum particles to perform unstructured search on a database of size n with a high success probability, in query complexity $\mathcal{O}(\sqrt{n})$. Grover's algorithm is also considered to be tight for unstructured search in the quantum realm, i.e., any other unstructured search algorithm requires to make at least $\sqrt{(n)}$ queries. It is important to note that Grover's algorithm works in the quantum realm, i.e., relies on the existence of efficient quantum computers. We are still pretty far from building a practical functioning quantum computer because of the existence of errors in quantum communication [5].

ResolveSat

ResolveSat is a simple randomized algorithm for k-SAT in CNF which finds a satisfying assignment in $\mathcal{O}(2^{0.446n})$ time with a high probability. The given algorithm performs much better than Schóning's algorithm, however, relies on the fact that the k-SAT formula contains clauses which are disjunctions, i.e., it can not be generalized to constraint satisfaction problems with a simple analysis, unlike Schöning's [6].

Oghlan's Algorithm for k-SAT

Oghlan's Algorithm is special because it finds a satisfying assignment for k-SAT with high probability, in polynomial time. Schöning's algorithm runs in exponential time while ResolveSat runs in sub-exponential but super-polynomial time. Grover's algorithm is not compared to Oghlan's algorithm as Grover's runs in the quantum world while Oghlan's works for classical machines. Now, despite its phenomenal run time, Oghlan's can not be used for each and every k-SAT problem out there. Oghlan's algorithm relies on the constraint densities, i.e. $\frac{m}{n}$ where n is the number of variables in the problems and m is the number of clauses, being bounded in terms of k [7].

References

[1] T. Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 410–414, 1999.

[2] Russell Norvig. *Artificial Intelligence: A Modern Approach, 4th edition.* Pearson, 2020.

[3] Marc Renault. Four proofs of the ballot theorem. *Mathematics Magazine*, April 2007.

[4] R.B. Ash. *Information Theory.* Dover, 1965.

[5] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[6] PaturiRamamohan, PudlákPavel, E SaksMichael, and ZaneFrancis. An improved exponential-time algorithm for k-sat. *Journal of the ACM*, 2005.

[7] Amin Coja-Oghlan. A better algorithm for random k-sat. *SIAM J. Comput.*, 39(7):2823–2864, may 2010.