The Satisfiability Problem: Random Walk and Derandomization

Timo Eckstein

November 11, 2014

Abstract

This short paper should show you the nowadays two main approaches – randomized and deterministic algorithm- for solving satisfiability problems efficiently. Therefore we will have a closer look onto Uwe Schöning's algorithm which uses random walk, the attempt of Dantsin et al. to get it deterministic and the complete derandomization by Moser & Scheder. As this somehow a little more elaborate hand-out solely summarizes, simplifies as well as shortens the work of others, I have to emphasize that nearly every piece of the content is at the minimum an indirect quote of one reference.

1 Introduction

But we will start with getting a rough idea about the aforementioned algorithm, it stands reason to have some definitions before which were strongly resembling them Scheder implemented in his PhD thesis [3] and in the reference paper he published together with Moser [2].

Definition 1.1. *u literal* $\Leftrightarrow$ *u=x(variable) or* $u = \bar{x}$ *(negation of x)*

Definition 1.2. *A finite set C of literals over pairwise distinct variables is called a clause*

Definition 1.3. *A finite set of clauses is called a formula in CNF (Conjunctive Normal Form). CNFs which have no more than k distinguishing literals are denoted* ($\leq$ *k)-CNF formulas, ones with exactly k literals k-CNF formulas.*

Example 1.4. *For a* ($\leq$ *3)-CNF formula*

$$(\bar{a} \vee \bar{b} \vee \bar{c}) \wedge (a \vee c) \qquad\qquad a, b, c \in \{0, 1\}$$

Definition 1.5. *The task of deciding whether a CNF-formula F is satisfiable is labelled satisfiability problem (short: SAT). A SAT is satisfiable iff a assignment* $\alpha \in \{0,1\}^n$ *with number of literals n exists such that F(*α*)=1. Since infinite SAT are in general not solvable, you could only do research on finite SATs. They are called resembling to CNFs k-SAT.*

Definition 1.6. *A mapping* α*: V* $\rightarrow$ *{0,1} is called (truth) assignment, whereby V is the set of variables occurring in the formula.*

Remark 1.7. *Papadimitriou, who had introduced the idea of using random walk in 1991, proved with his quadratic algorithm the polynomial runtime of 2-SAT. For* $k \geq 3$ *k-SAT is NP-Complete. More precisely Stephen A.Cook established NP-completeness by discovering SAT as the first problem fulfilling the required conditions.*

2 Algorithm for solving k-SAT

2.1 brute-force and splitting

First of all it's kind of obvious that any satisfiability problem is solvable by just examining each possibility. Considering that an assignment F is satisfiable iff $F|_{[x\mapsto 1]}$ or $F|_{[x\mapsto 0]}$ is satisfiable, led to a inefficient recursive $O(2^n poly(n))$-algorithm which splits step-by-step the primary problem in parts of decreasing size until for every literal a value is chosen. Hence this brute-force algorithm is an upper bound for the runtime. Nevertheless especially for small k there are better runtimes possible. For instance for 3-SAT Rodošek achieved $O(1.476^n)$. Unhappily according to Scheder his algorithm based on splitting is far more complicated. .

2.2 Random Walk algorithm by Uwe Schöning[1]

2.2.1 The basic idea

Abstractly spoken the approach by Schöning is applying a Monte Carlo appraoch onto k-SAT. Both algorithm. The most important part with an eye toward the analysis and success is a restart after 3n steps. The probability that we do not find a satisfying assignment after t repetitions with independent random bits is

$$[1 - \mathbf{Pr}(N \leq 3n)]^t \leq e^{-\mathbf{Pr}(N\leq 3n)t}$$

Therefore, to achieve an acceptable error probability of, say, e^{-20} one needs to choose $t = \frac{20}{\mathbf{Pr}(N\leq 3n)}$ independent repetitions of Schöning's algorithm.

2.2.2 Pseudocode

After the basic idea is shown, the pseudo code will get presented. Because of the quite tight and still good intelligibility the notion of Scheder is used instead of the original ones of Schöning. As a result of that, we could easily take his notation and do not have to adapt it.

Schöning((≤ k)-CNF formula F)

1: $\alpha \overset{\text{u.a.r.}}{\leftarrow} \{0,1\}^n$ // sample α **u**niformly **a**t **r**andom
2: **return** Schöning-Walk(F, α)

Schöning-Walk((≤ k)-CNF formula F, assignment α)

1: **for** i = 0, .., t do
2: **for** i = 0, .., 3n do
3: **if** α satisfies F **then**
4: **return** α
5: **else**
6: C ← any clause of F unsatisfied by α
7: u $\overset{\text{u.a.r.}}{\leftarrow}$ C // a random literal from C
8: $\alpha \leftarrow \alpha[u \mapsto 1]$
9: **endif**
10: **endfor**
11: **return** failure
12: **endfor**

Schöning's quite simple algorithm works as follows:

- *At first an initial assignment is chosen, by picking 0 and 1 with the same probability for each literal.*
- *Then up to 3n local correction steps were performed. In a local correction step an arbitrary unsatisfied clause is selected out of which a literal is chosen uniformly at random. The value of this literal gets after that flipped whereby the picked clause gets satisfied.*
- *In the event of finding a satisfying assignment within these 3n steps, it's returned. Otherwise, failure is reported.*

2.2.3 Estimate of the runtime

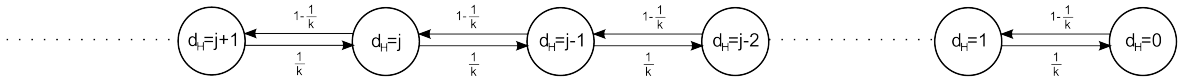
Now we finally get to – I would claim – most important feature of an algorithm, its runtime. The simple structure of Schöning's random walk algorithm is nice, but who cares if it takes more time for solving the same problem. But it is significant faster than the known splitting algorithm. For example the complexity of 3-SAT is given by $(\frac{4}{3})^n$.

Theorem 1. *Schöning's probabilistic algorithm which solves k-SAT problems is an* $O\Big(\Big(\frac{2(k-1)}{k}\Big)^n\Big)$*-algorithm.*

Remark 2.1. *Out of the given proofs of Schöning and Scheder, the one from Schöning using power series is chosen, because within it no polynomial terms occur. This is taken out of [1]:*

Proof

Consider the following corresponding infinite Markov chain, instead of the finite initial problem.: Let N_j *the random variable that counts the number of steps until the first encounter*



of state 0, assuming that the process starts in state j, i.e. $Y_0 = j$ *(Notice that it is possible that the state 0 will never be reached).*

Lemma 2.2. *For* $q < \frac{1}{2}$ *and* $j \in \mathbb{N}_0$*:*

$$\mathbf{Pr}(N_j < \infty) = \left(\frac{q}{1-q}\right)^j$$

Proof *By the ballot theorem the number of walks of length 2i+j from j to 0 where the first encounter of 0 happens in the last step is* $\binom{2i+j}{i}\frac{j}{2i+j}$*. Hence,*

$$\begin{aligned}\mathbf{Pr}(N_j < \infty) &= \sum_{i=0}^{\infty}\binom{2i+j}{i}\cdot\frac{j}{2i+j}\cdot(1-q)^i\cdot q^{i+j} = q^j\cdot\sum_{i=0}^{\infty}\binom{2i+j}{i}\cdot\frac{j}{2i+j}\cdot(q\cdot(1-q))^i \\ &= q^j\cdot(B_2(q(1-q)))^j\end{aligned}$$

for $B_2(z)$ *being the generalized Binomial series defined by*

$$B_2(z) = \sum_i\binom{2i+1}{i}\cdot\frac{z^i}{2i+1} = \frac{1-\sqrt{1-4z}}{2z}$$

for which

$$(B_2(z))^r = \sum_i\binom{2i+r}{i}\cdot\frac{r}{2i+1}\cdot z^i$$

$\forall r \in \mathbb{N}_0$. *So*

$$\mathbf{Pr}(N_j < \infty) = q^j \cdot \left(\frac{1-\sqrt{1-4q+4q^2}}{2(1-q)q}\right)^j = q^j \cdot \left(\frac{1}{1-q}\right)^j$$

Lemma 2.3. *For $q < \frac{1}{2}$ and $j \in \mathbb{N}_0$ it holds:*

$$\mathbf{E}(N_j \mid N_j > \infty) = \frac{j}{1-2q}$$

Proof

$$\mathbf{E}(N_j \mid N_j < \infty) = \frac{1}{\mathbf{Pr}(N_j < \infty)} \cdot \sum_{i=0}^{\infty}(2i+j) \cdot \binom{2i+j}{i} \cdot \frac{j}{2i+j} \cdot (1-q)^i \cdot q^{i+j}$$

$$\overset{\text{Lemma2.2}}{=} j \cdot (1-q)^j \cdot \sum_{i=0}^{\infty}(2i+j) \cdot \binom{2i+j}{i} \cdot (q\cdot(1-q))^i = j \cdot (1-q)^j \cdot \frac{(B_2(q(1-q)))^j}{\sqrt{1-4q(1-q)}} = \frac{j}{1-2q}$$

Lemma 2.4. *Let N the random variable that counts the number of steps until state 0 is encountered for the first time. For $q < \frac{1}{2}$ it holds, while n is still the number of variables:*

$$\mathbf{Pr}(N < \infty) = \left(\frac{1}{2(1-q)}\right)^n$$

Proof

$$\mathbf{Pr}(N < \infty) = \sum_{j=0}^{n}\binom{n}{j} \cdot 2^{-n} \cdot \mathbf{Pr}(N_j < \infty) \overset{\text{Lemma2.2}}{=} \sum_{j=0}^{n}\binom{n}{j} \cdot 2^{-n} \cdot \left(\frac{q}{1-q}\right)^j = \left(\frac{1}{2(1-q)}\right)^n$$

Lemma 2.5. *For $q < \frac{1}{2}$ it holds:*

$$\mathbf{E}(N \mid N < \infty) = \frac{qn}{1-2q}$$

Proof

$$\mathbf{E}(N \mid N < \infty) = \sum_i i \cdot \mathbf{Pr}(N = i \mid N < \infty) = \sum_i i \cdot \sum_{j=0}^{n}\binom{n}{j} \cdot \mathbf{Pr}(N_j = i \mid N < \infty)$$

$$= \frac{2^{-n}}{\mathbf{Pr}(N < \infty)} \cdot \sum_{j=0}^{n}\binom{n}{j} \cdot \sum_i i \cdot \mathbf{Pr}(N_j = i) = \frac{2^{-n}}{\mathbf{Pr}(N < \infty)} \cdot \sum_{j=0}^{n}\binom{n}{j} \cdot \mathbf{E}(N_j \mid N_j < \infty) \cdot \mathbf{Pr}(N_j < \infty)$$

$$\overset{\text{Lemma2.2,2.3,2.4}}{=} (1-q)^n \cdot \sum_{j=0}^{n}\binom{n}{j} \cdot \frac{j}{1-2q} \cdot \left(\frac{q}{1-q}\right)^j = \frac{n\cdot(1-q)^n}{1-2q} \cdot \sum_{j=0}^{n}\binom{n-1}{j-1} \cdot \left(\frac{q}{1-q}\right)^j$$

$$= \frac{n\cdot(1-q)^n}{1-2q} \cdot \frac{q}{1-q} \cdot \left(1+\frac{q}{1-q}\right)^{n-1} = \frac{nq}{1-2q}$$

Lemma 2.6. *For $q < \frac{1}{2}$ and $\lambda \geq 1$ it holds:*

$$\mathbf{Pr}\left(N \leq \frac{\lambda q n}{1-2q}\right) > \left(1-\frac{1}{\lambda}\right) \cdot \left(\frac{1}{2(1-q)}\right)^n$$

Proof *Write μ for $\mathbf{E}(N \mid N < \infty)$.Observe that*

$$\mathbf{Pr}(N > \lambda\mu \mid N < \infty) < \frac{1}{\lambda}$$

by Markov's inequality, and

$$\mathbf{Pr}(N \leq \lambda\mu) = \mathbf{Pr}(N > \lambda\mu \mid N < \infty) \cdot \mathbf{Pr}(N < \infty)$$

since $(N \leq \lambda\mu \wedge N < \infty) \Leftrightarrow (N \leq \lambda\mu)$ *. Now using* $q = \frac{1}{k}, k \geq 3$ *and* $\lambda = 3$*, we obtain*

$$\mathbf{Pr}(\exists t \leq 3n : Y_t = 0) = \mathbf{Pr}(N \leq 3n) > \frac{2}{3}\left(\frac{k}{2(k-1)}\right)^n$$

whereas, for a satisfiable formula, the expected number of repetitions of procedure Schöning until a satisfying assignment is found is at most $\frac{1}{\mathbf{Pr}(N \leq 3n)}$*:*

$$\frac{3}{2}\left(\frac{2(k-1)}{k}\right)^n$$

Remark 2.7. *By the time it was published Schöning's algorithm was the fastest algorithm for 3-SAT and was just slightly beaten by the more complex PPSZ-algorithm in the cases of* $k = 4$*, 5 and 6. But still today the fastest 3-SAT algorithms are barely faster (*$O(1.3298^n)$*) than the simple one from Schöning.*

2.3 Derandomization attempt by Dantsin et al. [4]

2.3.1 The basic idea

The main idea Moser and Scheder virtually accomplished originally came from Dantsin et al. ([4]). Although they derandomized Schöning's algorithm, they haven't managed to archive the same or a better runtime. The principle used is to cover the whole $\{0,1\}^n$ *space with Hamming balls of a fixed radius, named covering code.*

2.3.2 Pseudocode

For understanding that algorithm it was split by Scheder in two parts and each was considered separately. As his analysis is quite well-linked with the algorithm this version of that algorithm will be presented.Nevertheless before we could start we will need some more definitions.

Definition 2.8. $d_H(\alpha, \beta) := |x \in V | \alpha(x) \neq \beta(x)|$ *with* α, β *truth assignments is called Hamming distance*

Definition 2.9. $B_r(\alpha) := \{d_H(\alpha, \beta) \leq r\}$ *is denoted Hamming ball, with volume* $vol(n, r) := |B_\alpha(r)| = \sum_{i=0}^{r} \binom{n}{i}$*.*

Definition 2.10. *Ball-k-SAT: Decide whether* $B_r(\alpha)$ *contains a satisfying assignment*

Definition 2.11. $C \subseteq \{0,1\}^n$ *covering code of radius r and length* $n \Leftrightarrow \bigcup_{\alpha \in C} B_r(\alpha) = \{0,1\}^n$

In prinziple Dantsin et al's algorithm works as follows:

- *At first an initial covering code is created with a specific radius r. But in order to understand why these limits are taken, compared with the lemma and theorem of the next subsection*
- *Then the recursive function sat-searchball is called checking at the beginning whether the condition "*α *alpha satisfies F" and the trivial case r=0. Both together depict the exit condition.*
- *In the recursive call the radius is decreased by one by taking one literal and pursuing the algorithm under the constraint that this literal should be true.*

cover-search((≤ k)-CNF formula F over n variables)

1: $r := \frac{n}{k+1}$
2: construct a covering code C of radius r and $|C| \leq \frac{2^n}{\binom{n}{k}} poly(n)$
3: **return** $\bigcup_{\alpha \in C} sat - searchball(F, \alpha, r)$

$sat - searchball((\leq k)$-CNF formula F, assignment α, radius $r)$

1:**if** α satisfies F **then**
2: **return** true
3: **elseif** r = 0 **then**
4: **return** false
5: **else**
6: C ← any clause of F unsatisfied by α
7: **return** $\bigcup_{u \in C} sat - searchball(F|_{u:=1}, \alpha, r-1)$
8: **endif**

To see correctness of the algorithm, we proceed by induction on r.

Initial step: $r = 0 \Rightarrow B_0(\alpha) = \{\alpha\}$ vacuously true.
Induction step: Let be α^* a satisfying assignment with $d_H(\alpha, \alpha^*) \leq r$ and C the selected clause. Since α^* satisfies C but α does not, there is at least one literal $u \in C$ such that $\alpha^*(u) = 1$ and $\alpha(u) = 0$. Let $\alpha' := \alpha^*[u := 0]$. We observe that $d_H(\alpha, \alpha') \leq r - 1$ and $\alpha\prime$ satisfies $F|_{u:=1}$ (induction hypothesis).

2.3.3 Estimate of the runtime

Lemma 2.12. *The algorithm sat-searchball solves BALL-k-SAT in time $O(k^r poly(n))$.*

Proof *If F is a $(\leq k)$-CNF formula, then each call to searchball causes at most k recursive calls.*

Theorem 2. *Suppose some algorithm A solves BALL-k-SAT in $O(a^r poly(n))$ steps. Then there is an algorithm B solving k-SAT in time $O\Big(\Big(\frac{2a}{(a+1)}\Big)^n poly(n)\Big)$, and B is deterministic if A is.*

Proof

Lemma 2.13. *For all $n \in N$ $0 \leq r \leq n$, every code C of covering radius r and length n has at least $\frac{2^n}{vol(n,r)}$ elements. Furthermore, there is such a C with*

$$|C| \leq \frac{2^n poly(n)}{vol(n, r)},$$

and furthermore, C can be constructed deterministically in time O($|C|$ poly(n)).

Proof
This Lemma is the case k=2 of Lemma 2.19.

Lemma 2.14. *For $0 \leq \rho \leq \frac{1}{2}$ and $t \in \mathbb{N}$, it holds that*

$$\binom{t}{\rho t} \geq \frac{1}{\sqrt{8t\rho(1-\rho)}} \left(\frac{1}{\rho}\right)^{\rho t} \left(\frac{1}{1-\rho}\right)^{(1-\rho)t}$$

That proof is not such important for us, as it is a pure mathematical one, so it would not be explained in detail. According to Scheder you could read it up in [5].

Set $r := \frac{n}{(a+1)}$ and construct a covering code C of radius r and length n and call $A(F, \alpha, r)$ for each $\alpha \in C$.

$$|C|a^r poly(n) \overset{\text{Lemma2.13}}{\leq} \frac{2^n a^r poly(n)}{vol(n,r)} \overset{\text{Lemma2.14}}{\leq} \frac{2^n a^{\frac{n}{a+1}} poly(n)}{(a+1)^{\frac{n}{a+1}} \left(\frac{a+1}{a}\right)^{\frac{na}{a+1}}} = \left(\frac{2a}{a+1}\right)^n poly(n)$$

A more detailed calculation (using elementary calculus for example) also shows that the choice $r = \frac{n}{a+1}$ is indeed optimal.Since sat-searchball solves BALL-k-SAT in time $O(k^r poly(n))$, we can solve k-SAT in time $O\Big(\Big(\frac{2k}{(k+1)}\Big)^n poly(n)\Big)$. This guarantees Theorem 2. Scheder summarized this in the algorithm cover-search. As this runtime is larger than the one from Schöning's algorithm, it is not an equivalent derandomization.

Remark 2.15. *Not only to keep it easily unified but also because of the good structure and the concise mathematics the previous section is nearly entirely taken from Scheder's PhD thesis [3] and amplified with his paper with Moser [2], too.*

2.4 Complete Derandomization by Moser & Scheder [2]

2.4.1 The basic idea

Based on Dantsin et al.'s algorithm Schöning and Moser carried on and achieved the same runtime as Schöning with a far more complex but deterministic algorithm using more than two truth values. Additionally they had to make a constraint of Ball-k-SAT, they denoted Promise-Ball-k-SAT. Altogether their algorithm is more a further development of Dantsin et al.'s covering code idea led by the will to get an equivalent derandomised algorithm.

Definition 2.16. *Promise-BallL-k-SAT. Given a ($\leq$ k)-CNF formula F, an assignment α and a radius r, Promise-Ball-k-SAT is the following promise problem: If $B_r(\alpha)$ contains a satisfying assignment for F, answer true; if F is unsatisfiable, answer false; otherwise, i.e., if F is satisfiable but $B_r(\alpha)$ contains no satisfying assignment, answer true or false, arbitrarily.*

2.4.2 Pseudocode

The preceding definition for the Hamming distance also works for k truth values because whether only matters if two coordinates distinguish from each other. Consequently also the Hamming ball definition holds.

Definition 2.17. *$vol^{(k)}(t,r) := |B^{(k)}(w)| = \binom{t}{r}(k-1)^r$, $w \in \{1,...,k\}^t$. This is well-definied, because there are $\binom{t}{r}$ possibilities to pick a the set of coordinates in which w and $w' \in \{1,\ldots,k\}^t$ are supposed to differ, and for each such coordinate, there are k - 1 ways in which they can differ.*

Definition 2.18. *Let $t \in \mathbb{N}$. A set $C \subseteq \{1,..,k\}^t$ is called k-ary covering code of radius $r \Leftrightarrow \bigcup_{w \in C} B_r^{(k)}(w) = \{1,..,k\}^t$*

$searchball - fast(k \in \mathbb{N}$, ($\leq$ k)-CNF formula F, assignment α, radius r, code C$\subseteq \{1,...,k\}^t$)
1:**if**α satisfies F **then** 2: **return** true 3: **elseif** r = 0 **then** 4: **return** false 5: **else** 6: G $\leftarrow$ a maximal set of pairwise disjoint k-clauses of F unsatisfied by α 7: **if**$\lvert G\rvert < t$ **then** 8: **return** $\bigcup_{\beta\in\{0,1\}^{vbl(G)}} sat - searchball(F\lvert_{[\beta]}, \alpha, r)$ 9: **else** 10: H $\leftarrow \{C_1,...,C_t\} \subseteq$ G 11: **return** $\bigcup_{w\in C} searchball - fast(k, F, \alpha[H,w], r - (t-2t/k), C)$ 12: **endif** 13: **endif**

2.4.3 Estimate of the runtime

Just like some definitions we need also to generalize Lemma 2.13.

Lemma 2.19. *$\forall t,k \in \mathbb{N}$ and $0 \leq s \leq \frac{t}{2}$. $\exists$ $C \subseteq \{1,...,k\}^t$ of covering radius s such that:*

$$|C| \leq \left\lceil \frac{\ln(k)\cdot k^t \cdot t}{\binom{t}{s}\cdot(k-1)^s} \right\rceil =: m$$

and furthermore, C can be constructed deterministically in time O($|C|$ poly(t)).

Proof *Build C by sampling m points from {1,...,k}, uniformly at random and independently. Fix $w\prime \in \{1,\ldots,k\}^t$. Scheder and Moser than calculated:*

$$\mathbf{Pr}\left[w' \notin \bigcup_{w\in C} B_s^{(k)}(w)\right] = \left(1 - \frac{vol^{(k)}(t,s)}{k^t}\right)^{|C|} < e^{-|C|\frac{vol^{(k)}(t,s)}{k^t}} \leq e^{-t\cdot\ln(k)} = k^{-t}$$

By the union bound(= Boole's inequality), the probability that there is any $w\prime \notin \cup_{w\in C} B_s^{(k)}(w)$ is at most k^t times the above expression, and thus smaller than 1. Therefore, with positive probability, C is a code of covering radius s:= $\frac{t}{k}$.

Using the previous Lemma, remembering that k is constant and defining t:= $\lfloor ln(n) \rfloor$ Scheder shows a polynomial runtime of constructing such a covering code:

$$O(|C|\cdot poly(t)) \leq O(k^t \cdot poly(t)) = O(n^{\ln(k)} \cdot ploy(\ln(n))) = O(poly(n))$$

Lemma 2.20. *With the aid of Lemma 2.14 and Lemma 2.19 we get the following approximation of $|C|$ which we will need later. Let ρ be $\frac{1}{k}$:*

$$\binom{t}{\frac{t}{k}} \geq \frac{1}{\sqrt{8}} \cdot k^{\frac{t}{k}} \cdot \left(\frac{k}{k-1}\right)^{\frac{(k-1)\cdot t}{k}} = \frac{k^t}{\sqrt{8t} \cdot (k-1)^{\frac{(k-1)t}{k}}}$$

So we obtain, for t sufficiently large:

$$|C| \leq \left\lceil \frac{\ln(k) \cdot k^t \cdot t}{\binom{t}{s} \cdot (k-1)^s} \right\rceil \leq \frac{t^2 \cdot k^t \cdot (k-1)^{\frac{(k-1)t}{k}}}{k^t \cdot (k-1)^{\frac{t}{k}}} = t^2 \cdot (k-1)^{t-\frac{2t}{k}}$$

Now we eventually get around to the algorithm. At first the break conditions are implemented exactly like in the one before. Before it is splite in two cases a maximum set $G = \{C_1, ..C_m\}$ of pairwise disjoint unsatisfied k-clauses of F is constructed. If $m < t$ for all 2^{km} possible assignments β call sat-searchball($F|_\beta, \alpha, r$). This case is equal to the algorithm of Dantsin et al.. Since at least one β agrees with the promised assignment this is correct. For analysing the runtime we will need another lemma:

Lemma 2.21. *If every clause in F that is not satisfied by α has size at most k - 1, then sat-searchball(F, α, r) runs in time $O((k-1)^r poly(n))$.*

Proof *Since every unsatisfied clause and every $F|_{u:=1}$ has at most k - 1 literals, the algorithm calls itself at worst k – 1 times.*

Due to the maximality of G, $F|_\beta$ contains no unsatisfied clause of size k. Therefore this case takes:

$2^{km}O((k-1)^r poly(n)) \leq O(2^{kt}(k-1)^r poly(n)) \leq O(2^{ln(n)k}(k-1)^r poly(n)) \leq O((k-1)^r poly(n))$

Finally Theorem 2 provides that this part has a runtime equal to Schöning(F).

In case $m \geq t$ at first t clauses from G are picked out and named H. We define some functions for further use:

Definition 2.22. *For $H = \{C_1, \ldots, C_t\}$ and $w \in \{1, .., k\}^t$ let α[H,w] be the assignment obtained from α by flipping the value of the w_i^{th} literal in C_i, for 1≤ i ≤t. If H is understood write $\alpha[w]$ instead of $\alpha[H, w]$*

Definition 2.23. *Define $w^* \in \{1, \ldots, k\}^t$ as follows: for each 1≤ i ≤t set w_i^* to j such that α^* satisfies the j^{th} literal in C_i. This is possible since α^* satisfies at least one literal in each C_i. Then w_i^* is set to j.*

We could now call sat-searchball(F, α[w], r - t) for each $w \in 1, ..., k^t$. This would be no improvement over Dantsin et al. Instead, we iterate over all $w \in C$.

Observe:
- *There is some $w \in \{1, ..., k\}^t$ such that $d_H(\alpha[w^*], \alpha^*) = d_H(\alpha, \alpha^*) - t$*
- *Let w, $w' \in \{1, ..., k\}^t$. Then $d_H(\alpha[w], \alpha[w']) = 2d_H(w, w')$*

Hence we could prove following Lemma:

Lemma 2.24. *Let t and H be defined as above, and let $C \subseteq \{1, ..., k\}^t$ be a k-ary code of length t and covering radius s. If α^* is a satisfying assignment of F, then there is some $w \in C$ such that $d_H(\alpha[w], \alpha^*) \leq d_H(\alpha, \alpha^*)$ - t + 2s. In particular, if $B_r(\alpha)$ contains a satisfying assignment, then there is some $w \in C$ such that $B_{r-t+2s}(\alpha[w])$ contains it, too.*

Proof *C has covering radius $s \Rightarrow d_H(w, w^*) \leq s \overset{\text{Observation 2}}{\Rightarrow} d_H(\alpha[w], \alpha[w^*]) \leq 2s$*

$$d_H(\alpha[w], \alpha^*) \leq d_H(\alpha[w^*], \alpha^*) + d_H(\alpha[w], \alpha[w^*]) \leq d_H(\alpha, \alpha^*) - t + 2s$$

After the correctness of this algorithm has been shown the running time should be analysed. There are $|C|$ recursive calls and a decrease of the complexity parameter r by $t - 2s = t - \frac{2t}{k}$ in each step.Therefore:

$$|C|^{\frac{r}{t-\frac{2t}{k}}} \overset{\text{Lemma 2.20}}{\leq} \left(t^2(k-1)^{t-\frac{2t}{k}}\right)^{\frac{r}{t-\frac{2t}{k}}} = \left(t^{\frac{2}{t-\frac{2t}{k}}}(k-1)\right)^r = (k-1)^{r+o(n)}$$

Thereby the last step follows from the fact that $t^{\frac{2}{t-\frac{2t}{k}}}$ converges to 1 as t grows. This proves together with case one and Theorem 2 the complete derandomization of Schöning() which we will sum up in a theorem.

Theorem 3. *There is a deterministic algorithm solving k-SAT in time $O\left(\frac{2(k-1)}{k}\right)^{n+o(n)}$*

3 Conclusion

Finally I hope I could arouse at least a bit interest in k-SAT with this simplifying summary of [1], [2] and [4]. Nevertheless I have to stress again that seriously not negligible part is almost-copy from [3]. Additionally this paper is quite strong guided by Scheder's dissertation [3] which I would really recommend if you are interested in a more detail exposition.

References

[1] *U. Schöning:*
A probabilistic algorithm for k-SAT based on limited local search and restart; Algorithmica 32 (2002) 615-623

[2] *R.A. Moser, D. Scheder:*
A Full Derandomization of Schöninger's k-SAT Algorithm; in Proc. 43rd ACM Symp. On Theory of Computing (STOC), pp. 245-251, 2011

[3] *D. Scheder:*
Algorithms and Extremal Properties of SAT and CSP, dissertation, Swiss Federal Institute of Technology Zurich, 2011

[4] *E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P.Raghavan, U. Schöning:*
A deterministic $(2 - \frac{2}{k+1})^n$ algorithm for k-SAT based on local search. Theoretical Computer Science, Volumne 289, Issue 1, 23 October 2002, Page 69-83

[5] *F. J. MacWilliams and N. J. A. Sloane. The theory of error-correcting codes. II. North-Holland Publishing Co., Amsterdam, 1977. North-Holland Mathematical Library, Vol. 16.*