A Full Derandomization of Schöning's k-SAT Algorithm

Robin A. Moser
Institute for Theoretical Computer Science
Department of Computer Science
ETH Zürich, 8092 Zürich, Switzerland
robin.moser@inf.ethz.ch

Dominik Scheder
Institute for Theoretical Computer Science
Department of Computer Science
ETH Zürich, 8092 Zürich, Switzerland
dominik.scheder@inf.ethz.ch

ABSTRACT

Schöning [12] presents a simple randomized algorithm for k-SAT with running time $a_k^n \cdot \mathrm{poly}(n)$ for $a_k = 2(k-1)/k$. We give a deterministic version of this algorithm running in time $a_k^{n+o(n)}$.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*

General Terms

Algorithms

Keywords

Schöning's algorithm, derandomization, SAT, k-SAT

1. INTRODUCTION

In 1999, Uwe Schöning [12] gave an extremely simple randomized algorithm for k-SAT, which works as follows: Let F be an $(\leq k)$-CNF formula over n variables. Start with a random truth assignment. If this does not satisfy F, pick an arbitrary unsatisfied clause C. From C, pick a literal uniformly at random, and change the truth value of its underlying variable, thus satisfying C. Repeat this reassignment step $O(n)$ times. If F is satisfiable, this finds a satisfying assignment with probability at least

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

By repetition, this gives a randomized $O^*(1.334^n)$ algorithm for 3-SAT, $O^*(1.5^n)$ for 4-SAT, and so on (we use O^* to suppress polynomial factors in n). Actually this algorithm, also called WalkSAT, has been known before, see Selman, Kautz, and Cohen [13] for example. However, Schöning was the first to give a rigorous analysis of its success probability for $k \geq 3$. His analysis has its roots in a result by Papadimitriou [7], who proved that the expected running time of a suitable random walk algorithm is polynomial for 2-SAT.

Shortly after Schöning published his algorithm, Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan and Schöning [2] (henceforth *Dantsin et al.* for the sake of brevity) came up with a *deterministic* algorithm that can be seen as an attempt to derandomize Schöning's algorithm. We say *attempt* because its running time is $O^*((2k/(k+1))^n)$, which is exponentially slower than Schöning's. For example, this gives an $O^*(1.5^n)$ algorithm for 3-SAT and $O^*(1.6^n)$ for 4-SAT. Subsequent papers have improved upon this running time, mainly focusing on 3-SAT: Dantsin et al. already improve the running time for 3-SAT to $O(1.481^n)$, Brueggemann and Kern [1] to $O(1.473^n)$, Scheder [11] to $O(1.465^n)$, and Kutzkov and Scheder [5] to $O(1.439^n)$. All improvements suffer from two drawbacks: First, they fall short of achieving the running time of Schöning's randomized algorithm, and second, they are all fairly complicated. In this paper, we give a rather simple deterministic algorithm with a running time that comes arbitrarily close to Schöning's, thus completely derandomizing his algorithm. We also show how to derandomize Schöning's algorithm for constraint satisfaction problems, which are a generalization of SAT, allowing more than two truth values.

It should be noted that Schöning's algorithm is, by today, not anymore the fastest known algorithm for k-SAT. Instead, the PPSZ algorithm, named after its inventors Paturi, Pudlák, Saks and Zane [9], which has always known to be faster than Schöning for $k > 4$, has very recently been discovered to beat the running time of Schöning for all k by Timon Hertli in [3]. In particular, he showed that for $k = 3$, it has an expected running time of $O(1.30704^n)$. This has already been known to hold for the special case of 3-CNF formulas with a unique satisfying assignment. However, since both the algorithm and its analysis are far more complicated than Schöning's, derandomizing any of these algorithms will take far more work. First attempts at this have been made by Rolf in [10], but his derandomization only applies to formulas that come with the promise of having a unique satisfying assignment if any. Therefore, although bested within the randomized domain, the variant of Schöning's algorithm that we present hereafter is the fastest deterministic k-SAT algorithm known to date.

We think that derandomization of exponential algorithms is important. A recent result by Ryan Williams [15] has shown that fast deterministic exponential algorithms (run-

STOC'11, June 6–8, 2011, San Jose, California, USA.

ning in time $2^{0.99n}$, say) for the CircuitSAT problem would imply superpolynomial circuit lower bounds for NEXP, a nagging and long-standing open problem. On the other hand, Paturi and Pudlák [8] have shown that a polynomial-time probabilistic algorithm for CircuitSAT with success probability of at least $2^{-0.99n}$ can be sped up to a probabilistic algorithm with polynomial running time and success probability at least $2^{-0.01n}$. However, neither result's precondition is known to imply the other result's conclusion. It seems that both good deterministic and good probabilistic exponential algorithms can have very strong consequences. Therefore, we think it is important to try to translate one into the other.

1.1 Notation

We adapt the notational framework used in [14]. We assume an infinite supply of propositional *variables*. A *literal* u is a variable x or a complemented variable $\bar{x}$. A finite set C of literals over pairwise distinct variables is called a *clause* and a finite set of clauses is a *formula* in CNF (Conjunctive Normal Form). We say that a variable x *occurs* in a clause C if either x or $\bar{x}$ are contained in it and that x occurs in the formula F if there is any clause where it occurs. We write $\mathrm{vbl}(C)$ or $\mathrm{vbl}(F)$ to denote the set of variables that occur in C or in F, respectively. We say that F is a $(\leq k)$-*CNF formula* if every clause has size at most k. Let such an F be given and write $V := \mathrm{vbl}(F)$.

A *(truth) assignment* is a function $\alpha : V \to \{0,1\}$ which assigns a Boolean value to each variable. By $\{0,1\}^V$ we denote the set of all assignments to V. A literal $u = x$ (or $u = \bar{x}$) is *satisfied by* α if $\alpha(x) = 1$ (or $\alpha(x) = 0$). A clause is *satisfied by* α if it contains a satisfied literal and a formula is *satisfied by* α if all of its clauses are. A formula is *satisfiable* if there exists a satisfying truth assignment to its variables.

If α and β are two truth assignments over a set V of variables, then their *(Hamming) distance* $d_H(\alpha, \beta)$ is defined to be the number of variables $x \in V$ where $\alpha(x) \neq \beta(x)$, i.e. $d_H(\alpha, \beta) := |\{x \in V \mid \alpha(x) \neq \beta(x)\}|$. For a given assignment α, we denote the set of all assignments β with Hamming distance at most r from α by $B_r(\alpha) := \{\beta : V \to \{0,1\} \mid d_H(\alpha, \beta) \leq r\}$ and call this the *Hamming ball of radius r centered at α*.

Formulas can be manipulated by permanently assigning values to variables. If F is a given CNF formula and $x \in \mathrm{vbl}(F)$, then assigning $x \mapsto 1$ satisfies all clauses containing x (irrespective of what values the other variables in those closes are possibly assigned later) whilst it truncates all clauses containing $\bar{x}$ to their remaining literals. We will write $F^{[x:=1]}$ to denote the formula arising from doing just this, or equally $F^{[u:=1]}$ where u is a literal and we mean to assign the underlying variable the value necessary to satisfy u. If β is a partial assignment, i.e., defined on a subset of $\mathrm{vbl}(F)$, then $F^{[\beta]}$ denotes the formula we obtain from F by permanently setting the variables from those subset to their respective values under β.

1.2 Previous Work

Both Schöning's algorithm and its deterministic versions can be seen as not attacking SAT directly, but rather a parametrized local search problem:

> **Promise-Ball-k-SAT:** Given a $(\leq k)$-CNF formula F over n variables, an assignment α to these variables, a natural number r, and the promise that the Hamming ball $B_r(\alpha)$ contains a satisfying assignment. Find any satisfying assignment to F.

Let us clarify what we mean by saying "Algorithm A solves Promise-Ball-k-SAT": If F, α, and r are as described above, i.e., if $B_r(\alpha)$ contains a satisfying assignment, then A must return *some* satisfying assignment. We do not require this assignment to lie in $B_r(\alpha)$, however. On the other hand, if F is unsatisfiable, or $B_r(\alpha)$ contains no satisfying assignment, the behavior is unspecified. Of course, since we can quickly check any purported assignment that the algorithm outputs, we can assume the algorithm always either returns a satisfying assignment or `failure`.

The randomized algorithm for Promise-Ball-k-SAT originally used by Schöning as described in the introductory section, henceforth called `Schöning`, repeatedly selects any clause unsatisfied under α, then randomly picks a literal from that clause and flips the underlying variable's value. The algorithm gives up if a satisfying assignment has not been encountered by the time $O(n)$ steps have been performed (it is well-known and easy to check that $n/(k-2)$ correction steps are sufficient to achieve optimal efficiency).

Lemma 1 (Schöning [12]). *Let F be a $(\leq k)$-CNF formula, α a truth assignment to its variables, and $r \in \mathbb{N}$. If there is a satisfying assignment in $B_r(\alpha)$, then with probability at least $(k-1)^{-r}$,* `Schöning` *returns a satisfying assignment. By repetition, this gives a Monte-Carlo algorithm for* Promise-Ball-k-SAT *with running time $O^*((k-1)^r)$.*

Schöning turns this lemma into an algorithm for k-SAT by choosing the assignment α uniformly at random from all 2^n truth assignments:

Theorem 2 (Schöning [12]). *There is a randomized algorithm that runs in polynomial time and finds a satisfying assignment of F with probability*

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

provided F is satisfiable.

Proof. Let α^* be a satisfying assignment of F and let α be an assignment chosen uniformly at random from $\{0,1\}^n$. For each $0 \leq r \leq n$, the probability that the Hamming distance $d_H(\alpha, \alpha^*)$ is r is $\binom{n}{r}/2^n$. In this case, Schöning's random walk returns a satisfying assignment with probability at least $(k-1)^{-r}$. The overall success probability thus is at least

$$\sum_{r=0}^{n} \binom{n}{r} 2^{-n} (k-1)^{-r} = \left(\frac{k}{2(k-1)}\right)^n ,$$

and the running time is clearly polynomial. □

By repeating the above algorithm, one obtains a Monte-Carlo algorithm for k-SAT of running time $O^*((2(k-1)/k)^n)$.

Deterministic Algorithms

What about deterministic algorithms? Dantsin et al. [2] give a simple recursive algorithm for Promise-Ball-k-SAT running in time $O^*(k^r)$: If α satisfies F, we are done. Otherwise, if $r = 0$, we can return `failure`. If $r \geq 1$ and α does

not satisfy F, we let C be an unsatisfied clause. There are at most k literals in C, thus there are at most k possibilities to locally change α so as to satisfy C. We recursively explore each possibility, decreasing r by 1 (see Algorithm 1 for the details). The other achievement of Dantsin et al. is to show how a deterministic algorithm for Promise-Ball-k-SAT can be turned into a deterministic algorithm for k-SAT:

Lemma 3 (Dantsin et al. [2]). *If algorithm A solves Promise-Ball-k-SAT in time $a^{r+o(r)}$, then there is an algorithm B solving k-SAT in time $\left(\frac{2a}{a+1}\right)^{n+o(n)}$. Furthermore, B is deterministic if A is.*

Their algorithm to prove the lemma constructs a so-called covering code $\mathcal{C} \subseteq \{0,1\}^n$ with the property that every assignment $\alpha \in \{0,1\}^n$ has a codeword $\gamma \in \mathcal{C}$ at a suitably small Hamming distance from α. Schöning's randomized selection of an initial assignment is turned deterministic by iterating through all codewords $\gamma \in \mathcal{C}$ and solving Promise-Ball-k-SAT around each of them. Provided that the formula is satisfiable, one choice of $\gamma \in \mathcal{C}$ will be sufficiently close to a satisfying assignment for the subsequent local search to succeed.

The recursive algorithm for Promise-Ball-k-SAT due to Dantsin et al. has running time $O^*(k^r)$. Therefore Lemma 3 gives a running time of $O^*((2k/(k+1))^n)$. For $k = 3$, clever branching rules have been designed to improve upon the $O^*(3^r)$ bound, leading to the respective improvements on deterministic running times mentioned in the first paragraph of this paper.

1.3 Our Contribution

Theorem 4. *There exists a deterministic algorithm which solves Promise-Ball-k-SAT in time $(k-1)^{r+o(r)}$.*

Combining this theorem with Lemma 3 proves our main theorem:

Theorem 5. *There is a deterministic algorithm solving k-SAT in time $\left(\frac{2(k-1)}{k}\right)^{n+o(n)}$.*

In Section 3, we show how to obtain a similar derandomization result for CSP problems with more than two truth values. Before jumping into technical details, let us sketch the main idea of our improvement for $k = 3$. Let F be a 3-CNF formula and α some assignment. Suppose F contains t pairwise disjoint clauses $C_1, \dots, C_t$, all of which are unsatisfied by α. We let Schöning's random walk algorithm process these clauses one after the other: In each clause C_i, it picks one literal randomly and satisfies it. Thus, of all 3^t possibilities to choose one literal in each C_i, it chooses one uniformly at random. Let α^* be an assignment satisfying F. With probability at least 3^{-t}, Schöning's random walk chooses in each C_i a literal that α^* satisfies. In this case, the distance from α to α^* decreases by t. However, with much bigger probability, roughly $2^{-t/3}$, the random walk chooses the "correct" literal in $2t/3$ clauses C_i and a "wrong" literal in the remaining $t/3$. In this case, the distance from α to α^* decreases by $t/3$. This is the power of Schöning's algorithm: It hopes to make a modest progress of $t/3$, which is much more likely than making a progress of t. Our key observation is that this choice of Schöning can be derandomized: There is a set of (roughly) $2^{t/3}$ choices of which literal to satisfy in each C_i, such that at least one of them makes a progress of at least $t/3$.

2. THE ALGORITHM

To begin with, we will formally state and analyze the recursive algorithm given by Dantsin et al. [2] which is known to solve Promise-Ball-k-SAT in time $O^*(k^r)$.

Algorithm 1 `searchball`(CNF formula F, assignment α, radius r)

```
1: if α satisfies F then
2:    return true
3: else if r = 0 then
4:    return false
5: else
6:    C ← any clause of F unsatisfied by α
7:    return ⋁_{u∈C} searchball(F^[u:=1], α, r − 1)
8: end if
```

Proposition 6. `searchball` *solves* Ball-k-SAT *in time* $O^*(k^r)$

Proof. The running time is easy to analyze: If F is a $(\leq k)$-CNF formula, then each call to `searchball` causes at most k recursive calls. To see correctness of the algorithm, we proceed by induction on r and suppose that α^* satisfies F and $d_H(\alpha, \alpha^*) \leq r$. Let C be the clause selected in line 6. 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 $d(\alpha, \alpha') \leq r - 1$ and α' satisfies $F^{[u:=1]}$ (although not necessarily F). Therefore the induction hypothesis ensures that the recursive call to `searchball`$(F^{[u:=1]}, \alpha, r-1)$ returns `true`. □

Proposition 7. *Suppose F is a $(\leq k)$-CNF formula, α a truth assignment to its variables, and $r \in \mathbb{N}$. If every clause in F that is unsatisfied by α has size at most $k-1$, then* `searchball`(F, α, r) *runs in time $O^*((k-1)^r)$.*

Proof. The key observation is that if all clauses in F that are not satisfied by α have at most $k-1$ literals, then the same is true for any formula of the form $F^{[u:=1]}$. Therefore, any call to `searchball` entails at most $k-1$ recursive calls. □

2.1 k-ary Covering Codes

Before explaining our algorithm, we make a combinatorial detour to k-ary covering codes, which will play a crucial role in our algorithm. The set $\{1, \dots, k\}^t$ looks similar to the Boolean cube $\{0,1\}^t$ in many ways. For example, it is endowed with a Hamming distance d_H: For two elements $w, w' \in \{1, \dots, k\}^t$, we define $d_H(w, w')$ to be the number of coordinates in which w and w' do not agree. We define balls:

$$B_r^{(k)}(w) := \{w' \in \{1, \dots, k\}^t \mid d_H(w, w') \leq r\} .$$

What is the volume of such a ball? Well, there are $\binom{t}{r}$ possibilities to choose the set of coordinates in which w and w' are supposed to differ, and for each such coordinate, there are $k-1$ ways in which they can differ. Therefore,

$$\mathrm{vol}^{(k)}(t, r) := |B_r^{(k)}(w)| = \binom{t}{r}(k-1)^r .$$

We are interested in the question how many balls $B_r^{(k)}(w)$ we need to cover all of $\{1, \dots, k\}^t$. Note that by symmetry, $w \in B_r^{(k)}(v)$ iff $v \in B_r^{(k)}(w)$ for any $v, w \in \{1, \dots, k\}^t$.

DEFINITION 8. *Let $t \in \mathbb{N}$. A set $\mathcal{C} \subseteq \{1,\dots,k\}^t$ is called a* code of covering radius r *if*

$$\bigcup_{w\in\mathcal{C}} B_r^{(k)}(w) = \{1,\dots,k\}^t \ .$$

In other words, for each $w' \in \{1,\dots,k\}^n$, there is some $w \in \mathcal{C}$ such that $d_H(w,w') \le r$.

The following lemma is an adaptation of a lemma by Dantsin et al. [2], only for $\{1,\dots,k\}^t$ instead of the Boolean cube $\{0,1\}^t$.

LEMMA 9. *For any $t,k \in \mathbb{N}$ and $0 \le r \le t$, there exists a code $\mathcal{C} \subseteq \{1,\dots,k\}^t$ of covering radius r such that*

$$|\mathcal{C}| \le \left\lceil \frac{t\ln(k)k^t}{\binom{t}{r}(k-1)^r} \right\rceil$$

PROOF. The proof is probabilistic. Let

$$m := \left\lceil \frac{t\ln(k)k^t}{\binom{t}{r}(k-1)^r} \right\rceil$$

and build $\mathcal{C}$ by sampling m points from $\{1,\dots,k\}$, uniformly at random and independently. Fix an element $w' \in \{1,\dots,k\}^t$. We calculate

$$\Pr[w' \notin \bigcup_{w\in\mathcal{C}} B_r^{(k)}(w)] = \left(1 - \frac{\mathrm{vol}^{(k)}(t,r)}{k^t}\right)^{|\mathcal{C}|} < \\ < e^{-|\mathcal{C}|\mathrm{vol}^{(k)}(t,r)/k^t} \le e^{-t\ln(k)} = k^{-t}.$$

By the union bound, the probability that there is any $w' \notin \bigcup_{w\in\mathcal{C}} B_r^{(k)}(w)$ is at most k^t times the above expression, and thus smaller than 1. Therefore, with positive probability, $\mathcal{C}$ is a code of covering radius r. □

2.2 A Deterministic Algorithm for Promise-Ball-k-SAT

We will now describe our deterministic algorithm. Recall that r is the covering radius r that we choose for the covering code of starting assignments. Now let $t = t(r)$ be a slowly growing function. For instance, $t(r) = \log\log r$ would do. Compute a code $\mathcal{C} \subseteq \{1,\dots,k\}^t$ of covering radius t/k. Since k is a constant and t is very slowly growing, we can afford to compute an optimal such code. We estimate its size using Lemma 9 and the following approximation of the binomial coefficient:

PROPOSITION 10 ([6]). *For $0 \le \rho \le 1/2$ and $t \in \mathbb{N}$, it holds that*

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

We apply this bound with $\rho = 1/k$:

$$\binom{t}{t/k} \ge \frac{1}{\sqrt{8t}} k^{t/k} \left(\frac{k}{k-1}\right)^{(k-1)t/k} = \frac{k^t}{\sqrt{8t}(k-1)^{(k-1)t/k}} \ .$$

Together with Lemma 9, we obtain, for t sufficiently large:

$$|\mathcal{C}| \le \left\lceil \frac{t\ln(k)k^t}{\binom{t}{t/k}(k-1)^{t/k}} \right\rceil \le \\ \le \frac{t^2k^t(k-1)^{(k-1)t/k}}{k^t(k-1)^{t/k}} \le t^2(k-1)^{t-2t/k}.$$

The algorithm computes this code and stores it for further use. It then calls a recursive procedure that does the real stuff. That procedure first greedily constructs a maximal set G of pairwise disjoint unsatisfied k-clauses of F. That is, $G = \{C_1, C_2, \dots, C_m\}$, the C_i are pairwise disjoint, each C_i in G is unsatisfied by α, and each unsatisfied k-clause D in F shares at least one literal with some C_i.

At this point, the algorithm considers two cases. First, if $m < t$, it enumerates all 2^{km} truth assignments to the variables in G. For each such assignment β, it calls the subroutine `searchball`$(F^{[\beta]}, \alpha, r)$ and returns `true` if at least one such call returns `true`. Correctness is easy to see: At least one β agrees with the promised assignment α^*, and therefore α^* still satisfies $F^{[\beta]}$. To analyze the running time, observe that for any such β, the formula $F^{[\beta]}$ contains no unsatisfied clause of size k. This follows from the maximality of G. Therefore, Proposition 7 tells us that `searchball`$(F^{[\beta]}, \alpha, r)$ runs in time $O^*((k-1)^r)$, and therefore this case takes time $2^{km}O^*((k-1)^r)$. Since $m < t$, and $t \ll \log r$, this is $O^*((k-1)^r)$.

The second case is more interesting: If $m \ge t$, the algorithm chooses t clauses from G to form $H = \{C_1, \dots, C_t\}$, a set of pairwise disjoint k-clauses, all unsatisfied by α. At this point, our code will come into play, but first we introduce some notation: For $w \in \{1,\dots,k\}^t$, let $\alpha[w]$ be the assignment obtained from α by flipping the value of the $w_i{}^{\text{th}}$ literal in C_i, for $1 \le i \le t$. To do this, the algorithm has to choose a fixed but arbitrary ordering on H as well as on the literals in each C_i. Note that $\alpha[w]$ satisfies exactly one literal in each C_i, for $1 \le i \le t$. Strictly speaking $\alpha[w]$ depends not only on w, but also on H, so we should write $\alpha[H,w]$ instead of $\alpha[w]$. However, as long as H is understood, we write $\alpha[w]$.

Let us give an example. Suppose α is the all-0-assignment, $t = 3$ and $H = \{(x_1 \vee y_1 \vee z_1), (x_2 \vee y_2 \vee z_2), (x_3 \vee y_3 \vee z_3)\}$. Let $w = (2,3,3)$. Then $\alpha[w]$ is the assignment that sets y_1, z_2, and z_3 to 1 and all other variables to 0.

Consider now the promised satisfying assignment α^* with $d_H(\alpha,\alpha^*) \le r$. We define $w^* \in \{1,\dots,k\}^t$ as follows: For each $1 \le i \le t$, we set w_i^* to j such that α^* satisfies the j^{th} literal in C_i. Since α^* satisfies at least one literal in each C_i, we can do this, but since α^* possibly satisfies multiple literals in C_i, the choice of w^* is not unique. Note that in any case $d(\alpha[w^*],\alpha^*) = d(\alpha,\alpha^*) - t \le r - t$.

We could now iterate over all $w \in \{1,\dots,k\}^t$ and call `searchball`$(F, \alpha[w], r-t)$. This would essentially be what `searchball` does and would yield a running time of $O^*(k^r)$, i.e., no improvement over Dantsin et al. Therefore, we do not do this. Instead, we let our code $\mathcal{C}$ play its crucial role: Rather than recursing on $\alpha[w]$ for each $w \in \{1,\dots,k\}^t$, we recurse only for each $w \in \mathcal{C}$. By the properties of $\mathcal{C}$, there is some $w' \in \mathcal{C}$ such that $d_H(w',w^*) = t/k$. Observe what happens when we go from α to $\alpha[w']$: For at most t/k coordinates i, we have $w_i' \ne w_i^*$. For those coordinates, switching the $w_i'{}^{\text{th}}$ literal of C_i in the assignment α *increases* the distance to α^*. On the other hand, there are at least $t - t/k$ coordinates i where $w_i' = w_i^*$, and switching the $w_i'{}^{\text{th}}$ literal of C_i for such an i *decreases* the distance to α^*. We conclude that the distance increases at most t/k times and

decreases at least $t - t/k$ times. Therefore

$$d_H(\alpha[w'], \alpha^*) \leq d_H(\alpha, \alpha^*) + t/k - (t - t/k) \leq r - (t - 2t/k).$$

Writing $\Delta := (t - 2t/k)$, the procedure calls itself recursively with $\alpha[w]$ and $r - \Delta$ for each $w \in \mathcal{C}$ and at least one call will be successful. Let us analyze the running time: We cause $|\mathcal{C}|$ recursive calls and decrease the complexity parameter r by Δ in each step. This is good, since $|\mathcal{C}|$ is only slightly bigger than $(k-1)^{\Delta}$. We conclude that the number of leaves in this recursion tree is at most

$$|\mathcal{C}|^{r/\Delta} \leq (t^2(k-1)^{\Delta})^{r/\Delta} = \left((k-1)t^{2/\Delta}\right)^r \ .$$

Since $t^{2/\Delta}$ goes to 1 as t grows, the above term is at most $(k-1)^{r+o(n)}$. This proves Theorem 4. We summarize the whole procedure in `searchball-fast`.

Algorithm 2 `searchball-fast`($k \in \mathbb{N}$, ($\leq k$)-CNF formula F, assignment α, radius r, code $\mathcal{C} \subseteq \{1, \ldots, k\}^t$)

if α satisfies F **then**
  return `true`
else if $r = 0$ **then**
  return `false`
else
  $G \leftarrow$ a maximal set of pairwise disjoint k-clauses of F unsatisfied by α
  if $|G| < t$ **then**
    return $\bigvee_{\beta \in \{0,1\}^{\mathrm{vbl}(G)}}$ `searchball`$(F^{[\beta]}, \alpha, r)$
  else
    $H \leftarrow \{C_1, \ldots, C_t\} \subseteq G$
    return $\bigvee_{w \in \mathcal{C}}$ `searchball-fast`$(k, F, \alpha[H, w], r - (t - 2t/k), \mathcal{C})$
  end if
end if

3. CONSTRAINT SATISFACTION

Constraint Satisfaction Problems, short CSPs, are generalizations of SAT, allowing more than two truth values. Formally, suppose there is a set $V = \{x_1, \ldots, x_n\}$ of n variables, each of which can take on a value in $\{1, \ldots, d\}$. A *literal* is an expression of the form $(x_i \neq c)$ for $c \in \{1, \ldots, d\}$. A constraint is a disjunction of literals, for example

$$(x_1 \neq 7 \vee x_2 \neq 5 \vee x_3 \neq d) \ .$$

A CSP formula is a conjunction of constraints. We call it a $(d, \leq k)$-CSP formula if its variables can take d values and each constraint has at most k literals. An assignment α to the variables V is a function $\alpha : V \to \{1, \ldots, d\}$ and can be represented as an element from $\{1, \ldots, d\}^n$. We say α satisfies the literal $(x_i \neq c)$ if $\alpha(x_i) \neq c$. It satisfies a constraint if it satisfies at least one literal in it, and it satisfies a CSP formula if it satisfies each constraint in it. Finally, $(d, \leq k)$-CSP is the problem of deciding whether a given $(d, \leq k)$-CSP formula has a satisfying assignment. Note that $(2, k)$-CSP is the same as k-SAT. Also, $(d, \leq k)$-CSP is NP-complete except the following three cases: (i) $d = 1$, (ii) $k = 1$, (iii) $d = k = 2$. Cases (i) and (ii) are trivial problems, and (iii) is 2-SAT, which is solvable in polynomial time (well-known, not difficult to show, but still not trivial).

For the cases where $(d, \leq k)$-CSP is NP-complete, what can we do? Iterating through all d^n assignments constitutes an algorithm solving (d, k)-CSP in time $O^*(d^n)$. Schöning's algorithm [12] is much faster:

THEOREM 11 (SCHÖNING [12]). *There is a Monte-Carlo algorithm solving $(d, \leq k)$-CSP in time*

$$O^*\left(\left(\frac{d(k-1)}{k}\right)^n\right) \ .$$

Again, for $d = 2$ this is the running time of `Schöning` for k-SAT. In his original paper [12], Schöning describes how his algorithm seamlessly generalizes to arbitrary domain sizes $d \geq 2$: in each correction step, after a variable to reassign has been selected at random, another random choice is made among the $d-1$ values it may be changed to. The subsequent analysis in [12] also extends to this case.

However, there is a more direct way to reduce the $(d, \leq k)$-CSP for $d > 2$ to the Boolean problem which is then able to use any k-SAT algorithm as a black box: we simply select for each variable, uniformly at random and independently from the other variables, 2 out of the d possible values in the domain. Any satisfying assignment survives this restriction with probability exactly $(2/d)^n$ and thus any k-SAT algorithm with success probability p^n generalizes to a $(d, \leq k)$-CSP algorithm with success probability $(2p/d)^n$. When plugging in `Schöning` for k-SAT, we obtain Theorem 11.

In order to generalize our deterministic variant to arbitrary domain sizes, we will choose the simple route and derandomize the aforementioned reduction instead of trying to rework the whole analysis from the previous section, with the additional advantage that the result scales for any further improvement on the running time for deterministic k-SAT.

THEOREM 12. *There exists a deterministic algorithm having running time $O^*((d/2)^n)$ which takes any $(d, \leq k)$-CSP F over n variables and produces $l \in O^*((d/2)^n)$ Boolean $(\leq k)$-CNF formulas $\{G_i\}_{1 \leq i \leq l}$ such that F is satisfiable if and only if there exists some i such that G_i is satisfiable.*

Using the k-SAT algorithm we developed in the previous section, we then immediately get the derandomization of Theorem 11.

COROLLARY 13. *There is a deterministic algorithm solving $(d, \leq k)$-CSP in time*

$$\left(\frac{d(k-1)}{k}\right)^{n+o(n)} \ .$$

PROOF OF THEOREM 12. We start with a useful definition. A 2-*box* in $\{1, \ldots, n\}^d$ is a set of the form $B := P_1 \times \cdots \times P_n$, where $P_i \subseteq \{1, \ldots, d\}$ and $|P_i| = 2$. A 2-box can be seen as a subcube of $\{1, \ldots, d\}^n$ of side length 2 and full dimension. A *random* 2-box is a 2-box sampled uniformly at random from all 2-boxes in $\{1, \ldots, d\}^n$: This can be done by sampling each P_i independently, uniformly at random from all $\binom{d}{2}$ pairs in $\{1, \ldots, d\}$. As mentioned above, the probability that any fixed satisfying assignment of F lies within a random 2-box is $(2/d)^n$.

In order to derandomize this technique, we need to deterministically cover $\{1, \ldots, d\}^n$ with 2-boxes, in a fashion very similar to the covering codes used by Dantsin et al. [2]:

LEMMA 14. *Let $d, n \in \mathbb{N}$. There is a set $\mathcal{B}$ of 2-boxes in $\{1,\dots,d\}^n$ such that*

$$\bigcup_{D\in\mathcal{B}} B = \{1,\dots,d\}^n$$

and

$$|\mathcal{B}| \leq \left(\frac{d}{2}\right)^n \text{poly}(n) \ .$$

Furthermore, $\mathcal{B}$ can be constructed in time $O(|\mathcal{B}|)$.

Given this lemma, our algorithm is complete: It first constructs such a suitably small set $\mathcal{B}$ of 2-boxes, and then, for each 2-box $P_1 \times \cdots \times P_n = B \in \mathcal{B}$, outputs a $(\leq k)$-CNF formula arising from F by restricting the domain of the i^{th} variable to the values in P_i. This finishes the proof of the theorem. □

It remains to prove the lemma.

PROOF OF LEMMA 14. Note that if d is an even number, the proof is easy. For $1 \leq j \leq d/2$, define $P^{(j)} = \{2j-1, 2j\}$. Each element $w \in \{1,\dots,d/2\}^n$ defines the 2-box

$$B_w := P^{(w_1)} \times \cdots \times P^{(w_n)}$$

and clearly

$$\bigcup_{w\in\{1,\dots,d/2\}^n} B_w = \{1,\dots,d\}^n \ .$$

The difficulty arises if d is odd. As Dantsin et al. [2], we first show the existence of a suitable set of 2-boxes, and then use a block construction and an approximation algorithm to obtain a construction.

LEMMA 15. *For any $n, d \in \mathbb{N}$, there is a set $\mathcal{B}$ of 2-boxes such that $|\mathcal{B}| \leq \lceil n \ln(d)(d/2)^n \rceil$ such that $\bigcup_{B\in\mathcal{B}} = \{1,\dots,d\}^n$.*

PROOF. The proof works exactly like the proof of Lemma 9. We sample $\lceil n \ln(d)(d/2)^n \rceil$ many 2-boxes independently, uniformly at random and show that with positive probability, the resulting set has the desired properties. □

To prove Lemma 14, we have to derandomize the probabilistic argument we have just seen. For this, we choose a sufficiently large constant b, set $n' := n/b$ and construct an instance of SET-COVER: The ground set is $\{1,\dots,d\}^{n'}$ and the sets are all 2-boxes therein, of which there are $\binom{d}{2}^{n'} \leq d^{2n'}$. We know from Lemma 15 that there is a cover of 2-boxes of size $\lceil n \ln(d)(d/2)^n \rceil$. There is a greedy algorithm for SET-COVER (see Hochbaum [4] for example) achieving an approximation ratio of $O(\log N)$, where N is the size of the ground set. Since in our case $\log N = \log(d^{n'}) = O(n')$, this algorithm will give us a set $\mathcal{B}$ of 2-boxes covering $\{1,\dots,d\}^{n'}$ of size

$$|\mathcal{B}| \in O\left((n')^2 \left(\frac{d}{2}\right)^{n'}\right) \ .$$

How much time do we need to construct $\mathcal{B}$? The greedy algorithm is polynomial in the size of its instance, which is $O(d^{2n'})$, thus it takes time $O(d^{2Cn'})$ for some constant C. By choosing b large enough, we can make sure that $d^{2Cn'} = d^{2Cn/b}$ is smaller than $(d/2)^n$. Finally, we obtain a set of 2-boxes in $\{1,\dots,d\}^n$ by "concatenating" the boxes in $\mathcal{B}$: We identify a tuple $(B_1,\dots,B_b) \in \mathcal{B}^b$ with the 2-box $B_1 \times \cdots \times B_b$, and therefore $\mathcal{B}^b$ is a set of 2-boxes covering $\{1,\dots,n\}$, and

$$|\mathcal{B}|^b \leq O\left(\left((n')^2 \left(\frac{d}{2}\right)^{n'}\right)^b\right) =$$

$$= O\left(\left(\frac{n}{b}\right)^{2b} \left(\frac{d}{2}\right)^n\right) = \left(\frac{d}{2}\right)^n \text{poly}(n).$$

This is a set of 2-boxes covering $\{1,\dots,d\}^n$ of the desired size.

4. CONCLUDING REMARKS AND OPEN PROBLEMS

Exponential vs. Polynomial Space.

Dantsin et al. construct their almost optimal covering code using exponential space. By using a simpler version of their result (for the reader who is familiar with covering codes or has read the previous section: by using linearly many blocks of constant length, rather than constantly many of linear length), we can iterate through the whole covering code in polynomial space, and therefore our algorithm uses only polynomial space.

A Promise Problem.

Let us define a promise version of k-SAT:

> For $\gamma \geq 1$, we define the problem γ-promise-SAT: Let F be a CNF formula such that there exists some (unknown) assignment that in every clause $C \in F$ satisfies at least $|C|/\gamma$ literals. Find *some* satisfying assignment of F.

Note that any satisfiable $(\leq k)$-CNF formula is an instance of γ-promise-SAT for $\gamma = k$. The reader who is familiar with Schöning's random walk algorithm [12] will quickly see that it solves γ-promise-SAT in expected time $O((2(\gamma-1)/\gamma)^n \text{poly}(\text{length}(F)))$. We write $\text{poly}(\text{size}(F)))$, since without upper bounds on the clause length, the bitsize of F can be superpolynomial in the number of variables. In particular, Schöning's algorithm is polynomial if $\gamma \leq 2$.

Can we use our approach to give a deterministic algorithm of running time

$$(2\gamma/(\gamma+1))^{n+o(n)} \text{poly}(\text{length}(F))?$$

Maybe if additionally we bound the clause length from above by some constant? The case where we find a sufficiently large set of pairwise disjoint unsatisfied clauses can be treated as above, but we have no idea how to treat the other case, i.e., when `searchball-fast` calls `searchball` in Line 8.

APPENDIX

A. ACKNOWLEDGEMENTS

We thank our supervisor Emo Welzl for continuous support. The second author thanks Konstantin Kutzkov for the fruitful collaboration on [5].

B. REFERENCES

[1] T. Brueggemann and W. Kern. An improved deterministic local search algorithm for 3-SAT. *Theor. Comput. Sci.*, 329(1-3):303–313, 2004.

[2] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, O. Raghavan, and U. Schöning. A deterministic $(2 - 2/(k + 1))^n$ algorithm for k-SAT based on local search. In *Theoretical Computer Science 289*, pages 69–83, 2002.

[3] T. Hertli. 3-SAT Faster and Simpler - Unique-SAT Bounds for PPSZ Hold in General. *ArXiv e-prints*, Mar. 2011.

[4] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., Boston, MA, USA, 1997.

[5] K. Kutzkov and D. Scheder. Using CSP to improve deterministic 3-SAT. *CoRR*, abs/1007.1166, 2010.

[6] 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.

[7] C. H. Papadimitriou. On selecting a satisfying truth assignment (extended abstract). In *Proceedings of the 32nd annual symposium on Foundations of computer science*, SFCS '91, pages 163–169, Washington, DC, USA, 1991. IEEE Computer Society.

[8] R. Paturi and P. Pudlák. On the complexity of circuit satisfiability. In L. J. Schulman, editor, *STOC*, pages 241–250. ACM, 2010.

[9] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005.

[10] D. Rolf. Derandomization of ppsz for unique- -sat. In *SAT*, pages 216–225, 2005.

[11] D. Scheder. Guided search and a faster deterministic algorithm for 3-SAT. In *Proc. of the 8th Latin American Symposium on Theoretical Informatics (LATIN'08), Lecture Notes In Computer Science, Vol. 4957*, pages 60–71, 2008.

[12] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 410, Washington, DC, USA, 1999. IEEE Computer Society.

[13] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *DIMACS SERIES IN DISCRETE MATHEMATICS AND THEORETICAL COMPUTER SCIENCE*, pages 521–532, 1995.

[14] E. Welzl. Boolean satisfiability – combinatorics and algorithms (lecture notes), 2005. `http://www.inf.ethz.ch/~emo/SmallPieces/SAT.ps`.

[15] R. Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 231–240, New York, NY, USA, 2010. ACM.