Using a Skewed Hamming Distance to Speed Up Deterministic Local Search

Dominik Scheder
Theoretical Computer Science, ETH Zürich
CH-8092 Zürich, Switzerland
dscheder@inf.ethz.ch

Abstract. Schöning [13] presents a simple randomized algorithm for \((d, k)\)-CSP problems with running time \(\left(\frac{d(k-1)}{k}\right)^n\text{poly}(n)\). Here, \(d\) is the number of colors, \(k\) is the size of the constraints, and \(n\) is the number of variables. A derandomized version of this, given by Dantsin et al. [2], achieves a running time of \(\left(\frac{dk}{k+1}\right)^n\text{poly}(n)\), inferior to Schöning’s. We come up with a simple modification of the deterministic algorithm, achieving a running time of \(\left(\frac{d(k-1)}{k}\cdot\frac{k^d}{d^k}\right)^n\text{poly}(n)\). Though not completely eliminating the gap, this comes very close to the randomized bound for all but very small values of \(d\). Our main idea is to define a graph structure on the set of \(d\) colors to speed up local search.

1 Introduction

Constraint Satisfaction Problems, short CSPs, are a generalization of both boolean satisfiability and the graph \(k\)-colorability problem. A set of \(n\) variables \(x_1, \ldots, x_n\) is given, each of which can take a value from \([d] := \{1, \ldots, d\}\). The values \(1, \ldots, d\) are sometimes called the colors. Each coloring of the \(n\) variables, also called assignment, can be represented as an element of \([d]^n\). A literal is an expression of the form \((x_i \neq c)\) for some \(c \in [d]\). A CSP formula consists of a conjunction (AND) of constraints, where a constraint is a disjunction (OR) of literals. We speak of \((d, k)\)-CSP formula if each constraint consists of at most \(k\) literals. Finally, \((d, k)\)-CSP is the problem of deciding whether a given \((d, k)\)-CSP formula has a satisfying assignment.

In 1999, Uwe Schöning [13] came up with an extremely simple and elegant algorithm for \((d, k)\)-CSP: Start with a random assignment. If this does not satisfy the formula, pick an arbitrary unsatisfied constraint. From this constraint, pick a literal uniformly at random, and assign to its underlying variable a new value, again randomly. Repeat this reassignment step \(O(n)\) times, where \(n\) is the number of variables in the formula. If the formula \(F\) is satisfiable, we find a satisfying assignment with probability at least \(\left(\frac{1}{(d(k-1))}\right)^n\text{poly}(n)\). By repeating this procedure \(\left(\frac{k^d}{d^k}\right)^n\text{poly}(n)\) times, we obtain an exponential Monte Carlo algorithm for \((d, k)\)-CSP which we will call Schöning. Not long afterwards, in
2002, Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan and Schöning \cite{dantsin} designed a deterministic algorithm based on deterministic local search and covering codes. This algorithm, henceforth called det-search, can be seen as an attempt to derandomize Schöning’s random walk algorithm (actually the authors cover only the case $d = 2$, but everything nicely generalizes to higher $d$). I say attempt because its running time of $(dk/(k+1))^n/poly(n)$ is worse than that of Schöning.

Consider the following variant of det-search: Suppose $F$ is a $(d, k)$-CSP formula on $n$ variables, with $d = 2^\ell$ being a power of 2. Replacing every $d$-ary variable by $\ell$ boolean variables, we transform $F$ into a $(2, \ell k)$-CSP formula $F'$ over $\ell n$ variables. We solve $F'$ using the original algorithm det-search for the boolean case. A quick calculation shows that this already improves over the running time of $(dk/(k+1))^n/poly(n)$. This observation motivates a more systematic exploration of possible ways to speed up det-search. The main contribution of this paper is a modified det-search algorithm, which achieves a significantly better running time. Both Schöning and det-search work by locally exploring the Hamming graph on $[d]^n$, in which two assignments are connected by an edge if they differ on exactly one variable. We define a graph $G$ on the set $\{1, \ldots, d\}$ of colors, thus obtaining a different, sparser graph on $[d]^n$, the $n$-fold Cartesian product $G^\square n$: Two assignments are connected by an edge if they differ on exactly one variable, and on that variable, the two respective colors are connected by an edge in $G$. With $G = K_d$, this is the Hamming graph on $[d]^n$. Taking $G$ to be the directed cycle on $d$ vertices, it turns out that our modified deterministic algorithm has a running time of

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

For $d \geq 3$, this is significantly better than det-search and comes very close to Schöning except if $d$ is very small (in particular, we do not improve the case $d = 2$). We hope that future research will eventually lead to a complete derandomization. We compare running times for some values of $d$ and $k$ (ignoring polynomial factors in $n$):

| $(d, k)$ | Schöning | det-search | this paper |
|---------|----------|------------|------------|
| $(2, 3)$ | $1.334^n$ | $1.5^n$    | $1.5^n$    |
| $(3, 3)$ | $2^n$    | $2.25^n$   | $2.077^n$  |
| $(5, 4)$ | $3.75^n$ | $4^n$      | $3.754^n$  |

The case of $(2, k)$-CSP, commonly called $k$-SAT, has drawn most attention, in particular 3-SAT. For 3-SAT, Schöning and det-search achieve a running time of $O(1.334^n)$ and $O(1.5^n)$, respectively. This is still very close to the current records: By combining Schöning with a randomized algorithm by Paturi, Pudlák, Saks, and Zane \cite{paturi}, Iwama and Tamaki \cite{iwama} achieved a running time of $O(1.3238^n)$. Later, Rolf \cite{rolf} improved the analysis of their algorithm to obtain
the currently best bound of $O(1.32216^n)$. The algorithm det-search has been improved as well, first to $O(1.481^n)$ by the same authors, then to $O(1.473^n)$ by Brueggemann and Kern [1], and finally to the currently best deterministic bound of $O(1.465^n)$ by myself [12]. Though we do not improve the case $d = 2$ in this paper, we hope that better understanding of general $(d,k)$-CSP will lead to better algorithms for $k$-SAT, as well.

Another fairly well-investigated case is $k = 2$ with $d$ being large. In 2002, Feder and Motwani [3] adapted a randomized $k$-SAT algorithm by Paturi, Pudlák and Zane [10] to $(d,2)$-CSP, obtaining a running time of $(c_d d^n)$, with $c_d$ converging to $e^{-1}$ as $d$ grows. We see that the base of the exponential term is proportional to $d$. A certain growth of the base with $d$ seems inevitable: Recently, Traxler [14] showed that an algorithm solving $(d,2)$-CSP in time $a^n$, with $a$ being independent of $d$, could be used to solve $k$-SAT in subexponential time, i.e., $2^{o(n)}$. This would contradict the exponential time hypothesis [6].

Organization of this paper

In Section 2 we describe Schöning and det-search, and analyze the running time of the latter. Although most of the material of Section 2 is from [13] and [2], we chose to present it here in order to make the paper self-contained. In Section 3, we define a graph structure on the set of colors, which changes the notion of distance on the set $[d]_n$. Taking this graph to be the directed cycle on $d$ vertices yields a significant improvement. In In Section 4, we show that choosing this graph is optimal.

2 The Algorithms Schöning and det-search

Schöning's algorithm works as follows. We start with a random assignment, and for $O(n)$ steps randomly correct it locally. By this we mean choosing an arbitrary non-satisfied constraint $C$, then choosing a literal $(x \neq c) \leftarrow_{u.a.r.} C$ (where $\leftarrow_{u.a.r.}$ means choosing something uniformly at random), and randomly re-coloring $x$ with some $c' \leftarrow_{u.a.r.} [d] \setminus \{c\}$.

**Theorem 2.1 ([13]).** If $F$ is a satisfiable $(d,k)$-CSP formula on $n$ variables, then One-Schöning-Run returns a satisfying assignment with probability at least

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

By repeating One-Schöning-Run $(d(k-1)/k)^n\text{poly}(n)$, times, we find a satisfying assignment with high probability. This yields a randomized Monte-Carlo algorithm of running time $(d(k-1)/k)^n\text{poly}(n)$, which we call Schöning.

Let us describe the algorithm det-search from [2]. We define a parametrized version of CSP, called BALL-CSP. Given a $(d,k)$-CSP formula $F$, an assignment
Algorithm 1 One-Schönig-Run ($F: a (d,k)$)-CSP) formula

1: $\alpha \leftarrow \text{u.a.r.} [d]^n$
2: for $i = 1, \ldots, cn$ do
3: // $c$ is a constant depending on $d$ and $k$, but not on $n$
4: if $\alpha$ satisfies $F$ then
5: return $\alpha$
6: else
7: $C \leftarrow$ any constraint of $F$ unsatisfied by $\alpha$
8: $(x \neq c) \leftarrow \text{u.a.r.} C$ // a random literal from $C$
9: $c' \leftarrow \text{u.a.r.} [d] \setminus \{c\}$ // choose a new color for $x$
10: $\alpha \leftarrow \alpha[x := c']$ // change the coloring $\alpha$
11: end if
12: end for
13: return unsatisfiable

For $\alpha \in [d]^n$ and an $r \in \mathbb{N}_0$, does there exist a satisfying assignment $\beta$ such that $d_H(\alpha, \beta) \leq r$? Here,

$$d_H(\alpha, \beta) = |\{1 \leq i \leq n \mid \alpha(x_i) \neq \beta(x_i)\}|$$

is the Hamming distance, and

$$B_r^{(d)}(\alpha) := \{\beta \in [d]^n \mid d_H(\alpha, \beta) \leq r\}$$

is the Hamming ball of radius $r$ around $\alpha$. In other words, BALL-CSP asks whether $B_r(\alpha)$ contains a satisfying assignment. The algorithm searchball solves it in time $(k(d-1))^r \text{poly}(n)$. To show correctness, suppose the ball $B_r^{(d)}(\alpha)$

contains a satisfying assignment $\beta$, and let $C$ be a constraint not satisfied by

Algorithm 2 searchball(CSP formula $F$, assignment $\alpha$, radius $r$)

1: if $\alpha$ satisfies $F$ then
2: return true
3: else if $r = 0$ then
4: return false
5: else
6: $C \leftarrow$ any constraint of $F$ unsatisfied by $\alpha$
7: for $(x \neq c) \in C$ do
8: for $c' \in [d] \setminus c$ do
9: $\alpha' \leftarrow \alpha[x := c']$
10: if searchball($F, \alpha', r - 1$) = true then
11: return true
12: end if
13: end for
14: end for
15: return false
16: end if
\(\alpha\). At least one literal \((x \neq c) \in C\) is satisfied by \(\beta\), and in one iteration of the inner for-loop, the algorithm will change the assignment \(\alpha\) to \(\alpha'\) such that \(\alpha'(x) = \beta(x)\), and therefore \(d_H(\alpha', \beta) = d_H(\alpha, \beta) - 1 \leq r - 1\), and at least one recursive call will be successful. The running time of this algorithm is easily seen to be at most \((k(d - 1))^r \text{poly}(n)\), as each call causes at most \(k(d - 1)\) recursive calls (see Figure 1 for an illustration), and takes a polynomial number of steps itself.

\[
\begin{array}{c}
(x \neq 1) \lor (y \neq 2) \lor (z \neq 2) \\
\hline
x \neq 1 & x \neq 2 & x \neq 3 \\
y \neq 2 & y \neq 3 & y \neq 4 \\
z \neq 2 & z \neq 3 & z \neq 4
\end{array}
\]

Fig. 1. \texttt{searchball} branching on a constraint of a \((3, 3)\)-CSP formula.

### Covering Codes

How can we turn this algorithm into an algorithm for searching \([d]^n\) for a satisfying assignment? Suppose somebody gives us a set \(C \subseteq [d]^n\) such that

\[
\bigcup_{\alpha \in C} B^{(d)}(\alpha) = [d]^n,
\]

i.e. a code of covering radius \(r\). By calling \texttt{searchball}(\(F, \alpha, r\)) for each \(\alpha \in C\), we can decide whether \([d]^n\) contains a satisfying assignment for \(F\) in time

\[
|C|(k(d - 1))^r \text{poly}(n).
\]

(1)

By symmetry of the cube \([d]^n\), the cardinality of \(B^{(d)}(\alpha)\) does not depend on \(\alpha\), and we define \(\text{vol}^{(d)}(n, r) := |B^{(d)}(\alpha)|\). The following lemma gives bounds on the size of a covering code \(C\).

**Lemma 2.2 ([2]).** For all \(n, d, r\), every code \(C\) of covering radius \(r\) has at least 

\[
\frac{[d]^n}{\text{vol}^{(d)}(n, r)}
\]

elements. Furthermore, there is such a \(C\) with

\[
|C| \leq \frac{[d]^n}{\text{vol}^{(d)}(n, r)} \text{poly}(n),
\]

and furthermore, \(C\) can be constructed deterministically in time \(|C|\text{poly}(n)\).
This lemma, together with (1), yields a deterministic algorithm solving \((d, k)\)-CSP in time \(\frac{d^n}{\text{vol}^{(d)}(n, r)}(k(d - 1))^r\text{poly}(n)\), and we are free to choose \(r\). At this point, Dantsin et al. use the estimate \(\text{vol}^{(2)}(n, r) = \sum_{i=0}^{r} \binom{n}{i} \geq 2^n H(r/n)/\text{poly}(n)\), where \(H(x)\) is the binary entropy function (see MacWilliams, Sloane [8], Chapter 10, Corollary 9, for example), but we prefer to derive the bounds we need ourselves, first because the calculations involved are simpler, and second because our method easily generalizes to the volume of more complicated balls we will define in the next section. We use generating functions, which are a well-established tool for determining the asymptotic growth of certain numbers (cf. the book \textit{generatingfunctionology} [15]).

\textbf{Lemma 2.3.} For any \(n, d \in \mathbb{N}\) and \(x \geq 0\), there is an \(r \in \{0, 1, \ldots, n\}\) such that

\[\text{vol}^{(d)}(n, r) \geq \frac{1}{n+1} \frac{(1+(d-1)x)^n}{x^r}.\]

\textbf{Proof.} We write down the generating function for the sequence \(\left(\binom{n}{i}(d-1)^ix^i\right)_{i=0}^{n}: (1+(d-1)x)^n = \sum_{i=0}^{n} \binom{n}{i}(d-1)^ix^i\). This sum involves \(n+1\) terms, the maximum being attained at \(i = r\) for some \(i \in \{0, \ldots, n\}\). Thus \((1+(d-1)x)^n \leq (n+1)\binom{n}{r}(d-1)^rx^r\). Using \(\text{vol}^{(d)}(n, r) \leq \binom{n}{r}(d-1)^r\) and re-arranging terms yields the claimed bound. \(\Box\)

\textbf{Theorem 2.4.} There is a deterministic algorithm solving \((d, k)\)-CSP in time \(\left(\frac{dk}{k+1}\right)^n\text{poly}(n)\).

\textbf{Proof.} Choose \(x := (k(d-1))^{-1}\) and apply Lemma 2.3. The lemma gives us some \(r \in \{0, \ldots, n\}\). With this radius, the running time in (1) is at most

\[\frac{d^n}{\text{vol}^{(d)}(n, r)}(k(d - 1))^r\text{poly}(n) = \frac{d^n x^r (k(d - 1))^r}{(1+(d-1)x)^n}\text{poly}(n) = \left(\frac{d}{1+(d-1)\frac{1}{k(d-1)}}\right)^n\text{poly}(n) = \left(\frac{dk}{k+1}\right)^n\text{poly}(n).\]

\(\Box\)

Let us summarize the algorithm \texttt{det-search}: It first constructs a code of appropriate covering radius, then calls \texttt{searchball} for every element in the code. Its running time is larger than that of \texttt{Schöning}, since \(dk/(k+1) \geq d(k-1)/k\).

3 \textit{G-Distance, G-Balls, and G-searchball}

Let \([d]\) be the set of colors, and let \(G = ([d], E)\) be a (possibly directed) graph. For two colors \(c, c'\), we denote by \(d_G(c, c')\) the length of a shortest path from \(c\) to \(c'\) in \(G\). If \(G\) is directed, this is not necessarily a metric, and therefore we
rather call it a *distance function*. It gives rise to a distance function on \([d]^n\): For two assignments \(\alpha, \beta \in [d]^n\), we define

\[
d_G(\alpha, \beta) = \sum_{i=1}^{n} d_G(\alpha_i, \beta_i).
\]

This is the shortest-path distance on the \(n\)-fold Cartesian product \(G^\square n\). This distance induces the notion of balls \(B_r^{(G)}(\alpha) := \{\beta \in [d]^n \mid d_G(\alpha, \beta) \leq r\}\), and of *dual balls* \(\{\beta \in [d]^n \mid d_G(\beta, \alpha) \leq r\}\). If \(G\) is undirected, balls and dual balls coincide, and for \(G\) being \(K_d\), the complete undirected graph, \(d_G\) is simply the Hamming distance. If \(G\) is vertex-transitive (and possibly directed), the cardinality \(|B_r^{(G)}(\alpha)|\) does not depend on \(\alpha\), and we define \(\text{vol}^{(G)}(n, r) := |B_r^{(G)}(\alpha)|\). By double-counting, this is also the cardinality of dual balls. In particular, a vertex-transitive graph is *regular*. Let \(\delta\) denote the number of edges leaving each vertex in \(G\). As before, we define a parametrized problem: Given \(F\), \(\alpha\) and \(r\), does \(B_r^{(G)}(\alpha)\) contain a satisfying assignment? Algorithm 3, almost identical to Algorithm 2, solves this problem in time \((\delta k)^r \text{poly}(n)\).

**Algorithm 3** \(G\)-searchball(CSP formula \(F\), assignment \(\alpha\), radius \(r\))

1. if \(\alpha\) satisfies \(F\) then
   2. return true
3. else if \(r = 0\) then
   4. return false
5. else
6. \(C \leftarrow\) any constraint of \(F\) unsatisfied by \(\alpha\)
7. for \((x \neq c) \in C\) do
   8. for all \(c'\) such that \((c, c') \in E(G)\) do
   9. \(\alpha' \leftarrow \alpha[x := c']\)
10. if searchball\((F, \alpha', r - 1) = \text{true}\) then
11. return true
12. end if
13. end for
14. end for
15. return false
16. end if

### 3.1 Covering Codes, Again

Using the \(G\)-distance function instead of the Hamming distance also induces the notion of covering codes. As before, \(C \subseteq [d]^n\) is a code of covering \(G\)-radius \(r\) if

\[
\bigcup_{\alpha \in C} B_r^{(G)}(\alpha) = [d]^n.
\]
The following lemma generalizes Lemma 2.2 to arbitrary vertex-transitive graphs $G$ on $d$ vertices. The proof does not introduce any new ideas, and can be found in the appendix.

**Lemma 3.1.** Let $d \geq 2$, and let $G$ be a vertex transitive graph on $d$ vertices. For all $n$ and $0 \leq r \leq n$, every code $C$ of covering $G$-radius $r$ has at least $d^n/\text{vol}(G)(n,r)$ elements. Furthermore, there is such a $C$ with $|C| \leq \lceil d^n/\text{vol}(G)(n,r) \rceil \text{poly}(n)$, and $C$ can be constructed deterministically in time $|C|\text{poly}(n)$.

By calling $G$-searchball$(F, \alpha, r)$ for each $\alpha \in C$, we can solve $(d,k)$-CSP deterministically in time

$$d^n/\text{vol}(G)(n,r) \text{poly}(n),$$

where we are free to choose any vertex-transitive graph $G$ and any radius $r$. Let us reflect over (3) for a minute. Taking a graph with many edges results in balls of greater volume, meaning a smaller $C$ but spending more time searching each ball. Taking $G$ to be rather sparse has the opposite effect. What is the optimal graph $G$ and the optimal radius $r$?

### 3.2 Directed Cycles

Let us analyze the algorithm using $G = C_d$, the directed cycle on $d$ vertices. Clearly, $\delta = 1$, and therefore $G$-searchball runs in time $k^r$. This is as fast as we can expect for any strongly connected graph. What is $\text{vol}(C_d)(n,r)$?

**Lemma 3.2.** For any $n,d \in \mathbb{N}$, and $x \geq 0$, there is an $r \in \{0,\ldots,(d-1)n\}$ such that

$$\text{vol}(C_d)(n,r) \geq \frac{(1+x+\cdots+x^{d-1})^n}{x^r} \cdot \frac{1}{\text{poly}(n)}.$$

**Proof.** Define $T(n,s) := \text{vol}(C_d)(n,s) - \text{vol}(C_d)(n,s-1)$. This is the number of assignments having distance exactly $s$ from a fixed assignment $\alpha$. Also, it is the number of vectors $a \in \{0,\ldots,d-1\}^n$ with $\sum_{i=1}^n a_i = s$. Writing down its generating function, we see that $(1+x+\cdots+x^{d-1})^n = \sum_{s=0}^{(d-1)n} T(n,s)x^s$. For some $r \in \{0,1,\ldots,(d-1)n+1\}$ that maximizes $T(n,r)x^r$, we obtain

$$(1+x+\cdots+x^{d-1})^n = \sum_{s=0}^{(d-1)n} T(n,s)x^s \leq ((d-1)n+1)T(n,r)x^r$$

Solving for $T(n,r)$ proves the lemma.  \[\Box\]
We apply this lemma for $x = \frac{1}{k}$ and obtain a certain radius $r$, for which we construct a code $C$ of covering $G$-radius $r$. Combining Lemma 3.2 with (3), we obtain a running time of 

$$\frac{d^nk^rx^r}{(1 + x + \cdots + x^{d-1})^n}\poly(n) = \left(\frac{d(k-1)}{k} \cdot \frac{k^d}{k^d - 1}\right)^n\poly(n) ,$$

and we have proven our main theorem.

**Theorem 3.3.** For all $d$ and $k$, there is a deterministic algorithm solving $(d, k)$-CSP in time 

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

There is one issue we have consistently been sweeping under the rug. We proved Lemma 2.3 and Lemma 3.2, but never addressed the question what radius $r$ fulfills the stated bound. For the analysis this does not matter, since $r$ cancels out nicely. However, if we were to implement the algorithm, we would have to choose the right radius. This is not difficult: In Lemma 3.2, the correct $r$ is the one maximizing $T(n, r)x^r$, and $T(n, r)$ can be computed quickly using dynamic programming.

### 4 Optimality of the Directed Cycle

We will show that our analysis cannot be improved by choosing a different vertex-transitive graph $G$ or a different radius $r$. We ignore graphs that are not vertex-transitive because we have no idea on how to upper bounding the running time of $G$-searchball, not to speak of estimating the size of a good covering code.

Let $G$ be a vertex-transitive graph on $d$ vertices. For some vertex $u \in V(G)$, we denote by $d_i$ the number of vertices $v \in V(G)$ having $d_G(u, v) = i$. Since $G$ is finite, the sequence $d_0, d_1, \ldots$, eventually becomes 0. Denoting the diameter of $G$ by $s$, it holds that $d_i = 0$ for all $i \geq s$. If $G$ is connected (which we do not necessarily assume), the $d_i$ add up to $d$. Since $G$ is vertex-transitive, the $d_i$ do not depend on the vertex $u$. Clearly, $G$ is $d_1$-regular, and $G$-searchball runs in time $(d_1k)^\poly(n)$ on a $(d, k)$-CSP formula. How do we estimate $\vol_G(n, r)$? Again, we define $T(n, r) = \vol_G(n, r) - \vol_G(n, r - 1)$, i.e., the number of elements having distance exactly $r$ from some fixed $\alpha$. The $T_G(n, r)$ obey the recurrence 

$$T(n, r) = \sum_{i=0}^{s} d_i T(n - 1, r - i) .$$

This is easy to see: Fix $\alpha \in [d]^n$. How many $\beta$ are there such that $d_G(\alpha, \beta) = r$? Consider the first coordinates $\alpha_1$ and $\beta_1$. If $d_G(\alpha_1, \beta_1) = i$, then there are $d_i$ ways to choose $\beta_1$, and the distances at the remaining $n - 1$ positions add up to $r - i$. Some moments of thought reveal the following identity:

$$\left(\sum_{i=0}^{s} d_i x^i\right)^n = \sum_{i=0}^{sn} T(n, i)x^i$$
Before, we were interested in bounding \( \text{vol}^{(G)}(n, r) \) from below. Now we want to bound it from above, because we want to argue that any code \( C \subseteq [d]^n \) of covering radius \( r \) must necessarily be large, and the algorithm must be slow.

\[ \text{Lemma 4.1.} \quad \text{For any } n \in \mathbb{N}, \ r \in \{0, 1, \ldots, sn\} \text{ and any } x \in [0,1], \text{ it holds that } \]

\[ \text{vol}^{(G)}(n, r) \leq \frac{\left( \sum_{i=0}^{s} d_i x^i \right)^n}{x^r}. \]

**Proof.** \( \left( \sum_{i=0}^{s} d_i x^i \right)^n = \sum_{i=0}^{sn} T(n, i)x^i \geq \sum_{i=0}^{r} T(n, i)x^i \geq \sum_{i=0}^{r} T(n, i)x^r = x^r \text{vol}^{(G)}(n, r), \) and for the last inequality we needed that \( x \in [0,1], \) thus \( x^i \geq x^r \) for \( i \leq r. \) Re-arranging terms yield the claimed bound. \( \square \)

Clearly any code \( C \subseteq [d]^n \) with \( \bigcup_{a \in C} B^{(G)}_r(a) = [d]^n \) must satisfy

\[ |C| \geq \frac{d^n}{\text{vol}^{(G)}(n, r)}. \]

Since \( G\text{-searchball} \) takes time \( (kd_1)^r, \) the total running time is at least

\[ \frac{d^n}{\text{vol}^{(G)}(n, r)}(kd_1)^r \geq \frac{d^n x^r (kd_1)^r}{\left( \sum_{i=0}^{d_i} x^i \right)^n}, \]

where this inequality holds for all choices of \( x. \) Setting \( x = \frac{1}{kd_1}, \) we see that the running time is at least

\[ \frac{d^n}{\left( \sum_{i=0}^{d_i} k^{-i} d_1^{-i} \right)^n}. \]

In a \( d_1 \)-regular graph, the number of vertices at distance \( i \) from \( u \) can be at most \( d_1^i. \) In other words, \( d_i \leq d_1^i, \) and the above expression is at least

\[ \frac{d^n}{\left( \sum_{i=0}^{k-1} k^{-i} \right)^n}, \]

which, up to a polynomial factor, is the same as what we get for the directed cycle on \( d \) vertices.

\[ 5 \text{ Conclusion and Open Problems} \]

We can apply the same idea to Schöning’s algorithm: When picking a literal \( (x \neq c) \) uniformly at random from an unsatisfied constraint of \( F \) (see line 8 of \textit{One-Schöning-Run}), we choose a new truth value \( c' \) uniformly at random from the set \( \{c' \in [d] \mid (c, c') \in E(G)\}. \) With \( G = K_d, \) this is the original algorithm Schöning, and surprisingly, for \( G \) being the directed cycle, one obtains exactly the same running time \( (d(k-1)/k)^n \text{poly}(n). \) Since the analysis of this modified Schöning does not introduce any new ideas, we refer the reader to the appendix and to Andrei Giurgiu’s Master’s Thesis [4], which presents a general framework for analyzing random walk algorithms for SAT. Our main open problem is the following.
For which graph on \( d \) vertices does the modified One-Schöning-Run achieve its optimal success probability?

If we had to, we would guess that no graph can improve Schöning’s algorithm. Intuitively, it does not make sense to restrict the random choices the algorithm can make, because we have no further information on which choice might be correct. In the deterministic case, where every branch is fully searched, it seems to make more sense to restrict the choices of the algorithm, since this yields an immediate reduction in the running time of \( G\text{-searchball} \).

Acknowledgments

Thanks a lot to Emo Welzl and Robin Moser for fruitful and pleasant discussions.

References

1. T. Brueggemann and W. Kern. An improved local search algorithm for 3-SAT. Memorandum 1709, Department of Applied Mathematics, University of Twente, Enschede, 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. Feder and R. Motwani. Worst-case time bounds for coloring and satisfiability problems. J. Algorithms, 45(2):192–201, 2002.
4. A. Giurgiu. Random walk algorithms for SAT, 2009. Master’s Thesis, ETH Zürich.
5. D. S. Hochbaum, editor. Approximation algorithms for NP-hard problems. PWS Publishing Co., Boston, MA, USA, 1997.
6. R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity. J. Comput. System Sci., 63(4):512–530, 2001. Special issue on FOCS 98 (Palo Alto, CA).
7. K. Iwama and S. Tanaki. Improved upper bounds for 3-sat. In SODA ’04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, pages 328–328, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.
8. F. J. MacWilliams and N. J. A. Sloane. The theory of error-correcting codes. I. North-Holland Publishing Co., Amsterdam, 1977. North-Holland Mathematical Library, Vol. 16.
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. R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. Chicago J. Theoret. Comput. Sci., pages Article 11, 19 pp. (electronic), 1999.
11. D. Rolf. Improved bound for the PPSZ/Schöning-algorithm for 3-SAT. Electronic Colloquium on Computational Complexity (ECCC), (159), 2005.
12. 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.
13. 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.

14. P. Traxler. The time complexity of constraint satisfaction. In Third International Workshop on Parameterized and Exact Computation (IWPEC), Lecture Notes in Computer Science, Vol. 5018, pages 190–201, 2008.

15. H. S. Wilf. generatingfunctionology. A K Peters Ltd., Wellesley, MA, third edition, 2006.
A Schöning’s Algorithm With Directed Cycles

Can we apply the same idea to Schöning’s algorithm? When picking a literal \((x \neq c)\) uniformly at random from an unsatisfied constraint of \(F\) (see line 8 of One-Schöning-Run), we choose a new truth value \(c'\) uniformly at random from \([d] \setminus \{c\}\). We modify this algorithm as follows: Using a graph \(G\) with vertex set \([d]\), we choose the new color uniformly at random from the set \(\{c' \in [d] \mid (c, c') \in E(G)\}\). If \(G = K_d\), this is nothing new. What if \(G\) is the directed cycle? Let the \(d\) colors be 0, 1, . . . , \(d - 1\) and let the edges be \((i, i + 1)\) (addition taken modulo \(d\)). This means that we always change color \(c\) to color \(c + 1\). Let \(\beta\) be a fixed satisfying assignment and \(\alpha\) be the current (non-satisfying) assignment in the algorithm One-Schöning-Run. If \(\beta\) satisfies the literal \((x \neq c)\), i.e. \(\beta(x) \neq c\), then changing the color of \(x\) from \(c\) to \(c + 1\) decreases the distance from \(\alpha\) to \(\beta\) by 1. Otherwise, if \(\beta(x) = c\), then the distance from \(\alpha\) to \(\beta\) increases by \(d - 1\). If \(C\) is a constraint involving \(k\) literals and which is unsatisfied by \(\alpha\), then with probability at least \(1/k\) we choose a literal that is satisfied by \(\beta\), and decrease the distance by 1, and with probability at most \((k - 1)/k\), we choose a literal not satisfied by \(\beta\), increasing the distance by \(d - 1\). To analyze the algorithm, we define a Markov chain (see Figure 2). The states of the Markov chain are \(\mathbb{N}_0 \cup \{S\}\), with

\[
\begin{align*}
\frac{1}{k} & \quad \frac{k - 1}{k} \\
0 & \quad j - 1 & \quad j & \quad j + d - 1
\end{align*}
\]

Fig. 2. Part of the Markov Chain

\(S\) being a special starting state. The states \(j \in \mathbb{N}_0\) represent the distance from \(\alpha\) to some fixed satisfying truth assignment \(\beta\). The transition probabilities are as follows: For \(0 \leq j \leq (d - 1)n\), the probability \(p_{S,j}\) of going from \(S\) to \(j\) is \(\frac{T_G(n,j)}{d^n}\), where \(T_G(n,j) = \text{vol}^G(n,j) - \text{vol}^G(n,j - 1)\) is the number of assignments \(\alpha\) such that \(d_G(\alpha, \beta) = j\). When taking a step from \(S\) to some \(j\) according to the transition probabilities, \(j\) will be distributed exactly as \(d_G(\alpha, \beta)\) for \(\alpha \in \text{u.a.r.} \ [d]^n\). Furthermore, for \(j \geq 1\), \(p_{j,j-1} = \frac{1}{k}\), and \(p_{j,j+d-1} = \frac{k - 1}{k}\), and \(p_{0,0} = 1\). Here, we only sketch analysis of this Markov chain. For details, please see Giurgiu’s Master Thesis [4]. The probability of One-Schöning-Run finding a satisfying assignment is at least the probability of this Markov chain reaching state 0 after at most \(cn\) steps, with \(c\) being the constant in line 2 of One-Schöning-Run. As it turns out, the probability that we reach 0 in at most \(cn\) steps, conditioned on the event that 0 is reached at all, is rather high. Note that with positive (in fact, quite large) probability, we will never reach state 0. Therefore, to analyze the success probability of One-Schöning-Run, it suffices to lower bound the probability that our random walk eventually reaches 0. Let \(P_j\) be the probability that a random
walk starting in state $j$ eventually reaches 0. The $P_j$ obey the equation

$$P_j = \frac{1}{k} P_{j-1} + \frac{k-1}{k} P_{j+d-1} \,.$$  

(4)

Observe that if some $\lambda \in (0, 1)$ satisfies

$$\lambda = \frac{1}{k} + \frac{k-1}{k} \lambda^d ,$$  

(5)

then $P_j = \lambda^j$ satisfies (4). Here we would have to show that (5) has a unique “reasonable” solution for each $d$, and that $\lambda^j$ is in fact the unique solution to (4). We can compute the probability that we eventually reach 0:

$$\Pr[0 \text{ eventually reached}] = \sum_{j=0}^{(d-1)n} T_G(n, j) \lambda^j = \frac{1}{d^n} (1 + \lambda + \lambda^2 + \cdots + \lambda^{d-1})^n ,$$

since $(1 + x + x^2 + \cdots + x^{d-1})^n = \sum_{i=0}^{(d-1)n} T_G(n, i) x^i$. The above expression involves a geometric series and thus equals $\left(\frac{\lambda^{d-1}}{d(\lambda - 1)}\right)^n$. From (5) we learn that $\lambda^d = \frac{k\lambda - 1}{k-1}$, and plugging this into the previous expression yields

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

Now the probability that One-Schöning-Run finds a satisfying assignment is at least $\left(\frac{k}{d(k-1)}\right)^n \frac{1}{d^k(n)}$, and if we repeat it $\left(\frac{d(k-1)}{k}\right)^n \poly(n)$ times, we find a satisfying assignment with constant probability (if one exists). This is exactly the running time of Schöning’s algorithm we got before. Hence we see: Running Schöning with $G$ being $K_d$ or being the directed cycle makes no difference.

B Constructing the Covering Code

We show how to deterministically construct a code $C \subseteq [d]^n$ of covering radius $r$, i.e., $\bigcup_{n \in C} B_G^{(G)}(n) = [d]^n$, for $G$ being the directed cycle on $d$ vertices. The construction is just a generalization of the one in Dantsin et al. [2].

**Lemma B.1.** Let $G$ be the directed cycle on $d$ vertices. For any $n \in \mathbb{N}$ and $x \geq$, there is an $r \in \{0, \ldots, (d-1)n\}$ such that

$$\text{vol}(G)(n, r) \geq \frac{1}{(d-1)n+1} \frac{(1 + x + x^2 + \cdots + x^{d-1})^n}{x^r} ,$$

and there is a code $C \subseteq [d]^n$ of size at most

$$\frac{[d]^n x^r}{(1 + x + x^2 + \cdots + x^{d-1})^n \poly(n)}$$

which can be constructed deterministically in time $O(|C|)$. 
Proof. The proof idea is as follows: A probabilistic argument shows that a code \( C^* \) of claimed size exists (one obtains \( C^* \) by sampling random points in \( [d]^n \)), and then one invokes a greedy polynomial time approximation algorithm for the Set Cover problem (see [3], for example). This returns a code of size at most \( |C^*|\text{poly}(n) \). The problem is that this instance of Set Cover has a ground set of size \( d^n \) and \( d^n \) sets to choose from, thus the approximation algorithm will take at least \( d^n \) steps. As in Dantsin et al. [2], we solve this problem by partitioning our \( n \) variables into \( b \) blocks of length \( n/b \) each, where \( b \) is a constant, depending on \( d \) but not \( n \).

Let us be more formal. We first construct a covering code for \( [d]^{n/b} \). By Lemma 3.2, we know that for any \( x \geq 0 \), there is an \( r \in \{0, \ldots, (d-1)n/b\} \) such that

\[
\operatorname{vol}(G)(n/b, r) \geq \frac{1}{(d-1)n + 1} \frac{(1 + x + \cdots + x^{d-1})^{n/b}}{x^r}.
\]

Using this \( r \), we choose a set \( C^* \subseteq [d]^{n/b} \) by randomly sampling \( \frac{\ln(d^{n/b}d^{n/b})}{\operatorname{vol}(G)(n/b, r)} \) elements from \( [d]^{n/b} \), uniformly at random with replacement. This is only a feature of the proof – the sampling is not part of our deterministic construction. For any fixed \( \beta \in [d]^{n/b} \), it holds that

\[
P[\beta \notin \bigcup_{\alpha \in C^*} B_{r}^{(G)}(\alpha)] = \left( 1 - \frac{\operatorname{vol}(G)(n/b, r)}{d^{n/b}} \right)^{|C^*|} < e^{-\ln(d^{n/b})} = d^{-n/b}.
\]

By the union bound, we see that with non-zero probability, no assignment \( \beta \) is uncovered, and thus there exists a code \( C^* \) of desired size and covering radius \( r \). We construct an instance of Set Cover: The ground set is \( [d]^{n/b} \), and the set system consists of all \( B_{r}^{(G)}(\alpha) \) for \( \alpha \in [d]^{n/b} \). The deterministic polynomial-time approximation algorithm will in time \( \text{poly}(d^{n/b}) \) find a code \( C \subseteq [d]^{n/b} \) of size \( O(|C^*|n) \). We define \( C' \subseteq [d]^{n} \) by \( C' := C^b \), the \( b \)-fold Cartesian product. It is easy to see that

\[
\bigcup_{\alpha \in C'} B_{r}^{(G)}(\alpha) = [d]^{n}
\]

and

\[
|C'| = |C|^b \leq \frac{d^n}{x^{rb}(1 + x + \cdots + x^{d-1})^n \text{poly}(n)^b}.
\]

By choosing \( b \) large enough, although still constant, we can make sure that the running time of the approximation algorithm is at most \( |C'| \). This concludes the proof.

Actually the proof works as well for arbitrary vertex-transitive graphs, not only directed cycles, but the formulas become uglier.