Electronic Colloquium on Computational Complexity, Report No. 12 (2001)

Algorithms for SAT and Upper Bounds on Their Complexity

Evgeny Dantsin[1] Edward A. Hirsch[2]

Sergei Ivanov[3] Maxim Vsemirnov[4]

January 2001

ABSTRACT. We survey recent algorithms for the propositional satisfiability problem, in particular algorithms that have the best current worst-case upper bounds on their complexity. We also discuss some related issues: the derandomization of the algorithm of Paturi, Pudlák, Saks and Zane, the Valiant-Vazirani Lemma, and random walk algorithms with the "back button".

1. INTRODUCTION

The propositional satisfiability problem (SAT) is one of the most natural $\mathcal{NP}$-complete problems, and therefore its complexity is crucial for the computational complexity theory. Since SAT is $\mathcal{NP}$-complete, it is unlikely that SAT can be solved in polynomial time. However, it is still important to understand how much time is required to solve SAT, even if this amount is exponential: an algorithm solving SAT in time, say, $2^{n/1000}$ would be quite useful for many applications, e.g., for contemporary circuit design problems.

Research in SAT algorithms includes *experimental* study of their performance as well as *theoretical* study of their complexity. This survey is concerned with the theoretical aspect. We discuss some recent algorithms for SAT having non-trivial worst-case upper bounds on their complexity. We also discuss interesting related issues, for example, is it possible to find satisfying assignments for uniquely satisfiable instances of SAT faster than to find satisfying assignments for arbitrary satisfiable instances? The paper gives a more thourough view of the existing SAT algorithms that have the best current worst-case upper bounds. We survey families of such algorithms and give some clarifying illustrations and open problems.

The novelty of the paper consists of three parts: we simplify the derandomization of Satisfiability Coding Lemma [PPZ97], give two new proofs of the Valiant-Vazirani Lemma [VV86], and prove some results about random walk algorithms for SAT.

---

[1]Department of Computer Science, University of Manchester, and Steklov Institute of Mathematics at St. Petersburg. Web: `http://logic.pdmi.ras.ru/~dantsin`. Supported by grants from EPSRC, INTAS and RFBR.

[2]Steklov Institute of Mathematics at St. Petersburg. Web: `http://logic.pdmi.ras.ru/~hirsch`. Supported by INTAS Fellowship YSF 99-4044 and grants from NATO and RFBR.

[3]Steklov Institute of Mathematics at St. Petersburg. E-mail: `svivanov@pdmi.ras.ru`.

[4]Steklov Institute of Mathematics at St. Petersburg. E-mail: `vsemir@pdmi.ras.ru`. Supported by RFBR (grant no. 99-01-00113).

ISSN 1433-8092

Basic definitions. We consider algorithms for the problem of satisfiability of a Boolean formula in conjunctive normal form (*CNF*). A formula in CNF is the conjunction of clauses, a clause is the disjunction of literals, and a literal is a Boolean variable or its negation. The satisfiability problem (denoted by *SAT*) is classically formulated as a decision problem: given a Boolean formula F in CNF, output "Satisfiable" if there is a truth assignment to its variables satisfying every clause of F, otherwise output "Unsatisfiable". However, we will treat it as the problem of computing a satifying assignment: output a satisfying assignment (if any), otherwise output "Unsatisfiable". Clearly, the two versions of SAT are equivalent up to a polynomial factor, i.e., if one of them is solvable in time $T(F)$ then the other is solvable in time $\mathrm{poly}(|F|) \cdot T(F)$, where $|F|$ is the length of F. Througout this paper the notation $f(t) = \mathrm{poly}(t)$ means that there exists a polynomial P such that the inequality $|f(t)| \leq P(t)$ holds for any t. The use of $\mathrm{poly}(\cdot)$ is similar to that of $O(\cdot)$.

Throughout this paper, F denotes the input formula, $|F|$ denotes its length, i.e., the total number of occurrences of all variables, n denotes the number of variables. Each of our algorithms takes F as its input, and outputs either a satisfying assignment to its variables or the answer "Unsatisfiable".

In the framework of randomized algorithms, we consider *algorithms with one-sided, admissible probability of error*, i.e., the outputted satisfying assignment is always correct, and the answer "Unsatisfiable" is correct with probability at least $\frac{1}{\mathrm{poly}(|F|)}$. By repeating such an algorithm a polynomial number of times, one can decrease the probability of error so that it becomes less than any pre-determined constant.

We use $F[x]$ to denote the formula obtained from F by setting the value of the variable x to `true`, i.e., by deleting all clauses containing the unnegated x and deleting the literal $\neg x$ from the remaining clauses. The formula $F[\neg x]$ is defined similarly (the value of x is set to `false`). Assignments are represented as sets of literals. If a positive literal x belongs to an assignment A, it means that A sets x's value to `true`; if $\neg x \in A$, then A sets x's value to `false`. For an assignment $A = \{l_1, l_2, \ldots, l_t\}$, the formula $F[A]$ is defined as $F[l_1][l_2]\ldots[l_t]$. Note that an assignment may be *partial*, i.e., some of the variables may remain unassigned.

A *k-clause* is a clause consisting of exactly k literals. A formula in *k-CNF* is a formula containing only i-clauses for $i \leq k$. The satisfiability problem for formulas in k-CNF is denoted by *k-SAT*.

Splitting algorithms. Many of SAT algorithms use *splitting*. By a *splitting algorithm* we mean an algorithm that reduces the problem for the input formula F to the problem for polynomially many formulas $F_1, \ldots, F_p$. The reduction can be deterministic (make a recursive call for each of the F_i's) or randomized (take one of the F_i's at random). It is natural to divide contemporary splitting algorithms for SAT into two families: DPLL-like and PPSZ-like algorithms.

DPLL-like algorithms are based on the procedures described in the papers of Davis and Putnam [DP60] and Davis, Logemann and Loveland [DLL62]. Roughly speaking, such an algorithm replaces the input formula F by two formulas $F[x]$ and $F[\neg x]$ obtained by setting the value of some variable x to `true` and `false` respectively. Then the algorithm simplifies each of the obtained formulas and makes a recursive call for each of the simplified formulas. The main tool for the analysis of such algorithms are recurrent equations for the recursion tree. Using this tool, Dantsin [Дан81] and Monien and Speckenmeyer [MS85] gave first non-trivial upper bounds for k-SAT. This technique has a simple representation in terms of Kullmann and Luckhardt's *branching tuples* [Kul99, KL97]. We describe the general scheme of a DPLL-like algorithm and the use of branching tuples in Sect. 2. We also list some heuristics used in modern DPLL-like algorithms to simplify formulas and to choose variables for assigning `true` and `false`.

Another family of splitting algorithms consists of *PPSZ-like algorithms* suggested by Paturi, Pudlák, Saks, and Zane [PPSZ98, PPZ97]. Such algorithms differ from DPLL-like algorithms basically in two points: first, in the choice of variables for assigning the values (random choice in PPSZ-like algorithms) and, second, in methods of analysis. Unlike local analysis of DPLL-like algorithms, the analysis of PPSZ-like algorithms is based on global arguments, for example, how many variables are never used for recursive calls because they are eliminated during simplification process. These algorithms essentially use randomness, however, some kind of derandomization is possible. We survey these algorithms in Sect. 3, and concentrate on derandomization issues.

A formula that has only one satisfying assignment is called a *uniquely satisfiable formula*. Unique-SAT, i.e., the problem of finding a satisfying assignment for a uniquely satisfiable formula, is somewhat easier for PPSZ-like algorithms (or, at least, for their analysis). The best current bounds for Unique-SAT are better than in the general case. This contrasts with the famous lemma of Valiant and Vazirani [VV86], which says that the randomized complexities of SAT and Unique-SAT are polynomially related. In Sect. 4, we give two new simple proofs of this lemma and discuss the problem of its application in our context.

Local search algorithms. There is a large family of algorithms that solve SAT using *local search* (see, e.g., [GPFW00] for survey). A typical local search algorithm starts from an initial assignment and modifies it step by step, trying to come closer to a satisfying assignment. If no satisfying assignment is found after a certain number of steps, the algorithm generates another initial assignment and modifies it step by step again. The number of such attempts is limited; if all of them fail to find a satisfying assignment, then the algorithm terminates with the answer "Unsatisfiable".

Methods of modifying assignments may vary. For example, *greedy algorithms* (e.g., [KP92, SLM92]) choose a variable and flip its value in the current assignment so that some function of the assignment (e.g., the number of clauses it satisfies) increases as much as possible. Another method is used in *random walk algorithms* [Pap91]. Such an algorithm flips the value of a variable chosen at random from an unsatisfied clause. The complexity of random walk algorithms can be estimated using their connection with one-dimensional random walks. The main results on upper bounds for these algorithms are due to Papadimitriou [Pap91] and Schöning [Sch99]. As shown in [Pap91], 2-SAT can be solved by a random walk algorithm in polynomial time. The recent paper [Sch99] shows that k-SAT can be solved by a random walk algorithm in time $(2 - 2/k)^n$ up to a polynomial factor. For $k = 3$ this bound is $(4/3)^n$, which is the best known upper bound for 3-SAT algorithms.

We discuss random walk algorithms in Sect. 5. In particular, it is shown in this section that Papadimitriou's algorithm can be used to solve SAT for renamable Horn formulas in polynomial time. We also describe Schöning's algorithm and its derandomization [DGH+00, DGHS00]. Then we modify the notion of random walk algorithms using the recent approach of Fagin et al. [FKK+00] who introduced *Markov chains with the "back button"*. Namely, we define random walk algorithms with the "back button" and compare their complexity with the complexity of ordinary random walk algorithms. Random walk algorithms with the "back button" can be viewed as a kind of combination of the random walk approach and the splitting approach.

Related problems. Research in worst-case upper bounds for hard problems is not limited to SAT. Other $\mathcal{NP}$-complete problems (e.g., MAX-SAT [NR00], MAX-2-SAT, MAX-CUT [GHNR00], 3-Coloring [BE95], Maximim Independent Set [Bei00, Rob86]) also received much attention during the past years. However, the above mentioned papers use mostly DPLL-like methods, and it is interesting whether PPSZ-like or local search methods can bring new upper bounds for such problems.

It is known that some hard problems have limits of polynomial-time approximation unless $\mathcal{P} = \mathcal{NP}$ (see, e.g., [AL97]). For example, for MAX-3-SAT there is no polynomial-time algorithm (unless $\mathcal{P} = \mathcal{NP}$) that finds an assignment satisfying $\geq (\frac{7}{8}+\epsilon)M$ clauses, where M is the maximum possible number of simultaneously satisfiable clauses, and $\epsilon > 0$ is arbitrarily small. However, there are algorithms (see, e.g., [DGHK01, Hir00b]) that find such approximate solutions faster than the best current algorithms find exact solutions, the latter paper uses a random walk algorithm similar to [Pap91, Sch99].

Acknowledgements. The authors are grateful to Natalia Tsilevich for help in proving Theorem 5.5, and to Dima Pasechnik for fruitful discussions concerning Sect. 5.

2. DPLL-like algorithms

General scheme. The algorithms described by Davis and Putnam [DP60] and Davis, Logemann and Loveland [DLL62] are usually referred to as first algorithms for SAT. Most algorithms designed in the next forty years are based on the algorithms of [DLL62, DP60]. The modern view of such *DPLL-like algorithms* is as follows:

Procedure 2.1.

1. Simplify the input formula F, i.e., modify F into another formula G by using certain *transformation rules*.
2. If the satisfiability problem is trivial for G, return the answer.
3. Choose a variable v occurring in G using a certain *heuristic*. Construct the formulas $G[v]$ and $G[\neg v]$ and make a recursive call for each of them. If at least one of the recursive calls returns a satisfying assignment, update it by adding v or $\neg v$ respectively and return the result (the updating may also include changes caused by the use of transformation rules). Otherwise, return the answer "Unsatisfiable".

This procedure is thus parametrized by

1. The transformation rules for the simplification of formulas (the simplification is assumed to run in polynomial time).
2. The heuristic for choosing a variable for splitting (also in polynomial time).

A huge amount of various transformation rules and heuristics is known. A simple (but sometimes lengthy) method of analysis is given by the following observation.

Branching tuples. The execution of Procedure 2.1 can be represented by a *splitting tree*. Its root is labelled by a formula obtained by simplifying the input formula F. If a node of the tree is labelled by a formula G, then its two sons are labelled by formulas obtained by simplifying $G[v]$ and $G[\neg v]$. The leaves are labelled by trivial formulas (containing no variables).

Given a splitting tree, for each its node we can write a recurrent inequality for an upper bound $T(n)$ on the running time of Procedure 2.1. Trivially, one can write

$$T(n) \le 2 \cdot T(n-1) + \mathrm{poly}(|F|)$$

since splitting decreases the number of variables at least by one. Usually, it is possible to write a "better" inequality. Its "quality" depends on

1. The transformation rules.
2. The choice of a variable for splitting.
3. Syntactic properties of the formula labelling the node (clearly, 1 and 2 also have effect on these properties).

In general, we can replace a simple splitting

$$G \longrightarrow G[v],\ G[\neg v]$$

by something more complex, e.g.,

$$G \longrightarrow G[v,w],\ G[\neg v,w],\ G[v,\neg w],\ G[\neg v,\neg w].$$

If we can prove something special about a particular formula G, e.g., that unsatisfiability of formulas $G[v,\neg w]$ and $G[\neg v,w]$ implies unsatisfiability of formulas $G[v,w]$ and $G[\neg v,\neg w]$, we can use splittings like

$$G \longrightarrow G[v,\neg w],\ G[\neg v,w],$$

obtaining more and more complex recurrences.

A nice framework for dealing with such recurrences was developed by Kullmann and Luckhardt [Kul99, KL97]. Instead of a recurrent inequality, each node of a splitting tree gets a *branching tuple*. For example, if we are interested in the complexity with respect to the number of variables in the formula, the branching tuple is formed in the following way. Consider splitting

$$G \longrightarrow G_1, \ldots, G_d$$

of a formula with n variables to the formulas $G_1, \ldots, G_d$ with $n_1, \ldots, n_d$ variables respectively. Then the branching tuple $(t_1, \ldots, t_d)$ consists of arbitrary numbers $t_i \le n - n_i$. The corresponding *branching number* is the unique solution of

$$\sum_{i=1}^{d} x^{-t_i} = 1$$

on the interval $(0, +\infty)$. Then the running time of Procedure 2.1 is upper-bounded by $\text{poly}(|F|) \cdot \tau^n$, where τ is the largest of the branching numbers for all nodes of our tree. The 3-SAT bound $\text{poly}(|F|) \cdot 1.505^n$ of [Kul99] obtained in this way was the best among deterministic algorithms for six years[1].

Similarly, we can estimate the running time with respect to the number m of clauses in the input formula, with respect to the total number l of the occurrences of all variables, and with respect to any other "reasonable" measure of complexity of the input formula. The best current bounds with respect to m and with respect to l are 1.239^m and 1.074^l up to a polynomial factor [Hir00a]. These are valid for arbitrary formulas in CNF, not just in 3-CNF.

Transformation rules. We now list some transformation rules used in DPLL-like algorithms. We refer the interested reader to [Kul98, Kul00, KL97] for further details.

(2.1) **Unit clause elimination.** If F contains a clause consisting of the only literal l, set the value of l to `true`.

(2.2) **Pure literal.** If F contains a *pure* literal, i.e., a literal l such that its negation does not occur in F, set the value of l to `true`.

(2.3) **Resolution.** For a variable x of F, add to F all resolvents on x and remove from F all clauses containing x or its negation (the resolvent of two clauses $x \vee l_{11} \vee \ldots \vee l_{1s}$ and $\neg x \vee l_{21} \vee \ldots \vee l_{2t}$ on x is the clause $l_{11} \vee \ldots \vee l_{1s} \vee l_{21} \vee \ldots \vee l_{2t}$ if $l_{1i} \ne \neg l_{2j}$ for all i, j, and `true` otherwise).

(2.4) **Subsumption.** If F contains two clauses $C \subseteq D$, remove D.

(2.5) **Autarkness.** If there is a (partial) assignment A such that $F[A]$ does not contain clauses not occurring in F, replace F by $F[A]$.

(2.6) **Black-and-white literals.** Let P be some polynomial-time computable property of formulas and literals. Assume also that $P(F, x)$ ("x is a white literal, $\neg x$ is a black literal") and $P(F, \neg x)$ ("$\neg x$ is a white literal, x is a black literal") cannot hold simultaneously. If each clause of F that contains a literal l satisfying $P(F, l)$ also contains a literal l' satisfying $P(F, \neg l')$, replace F by $F[\{l' \mid P(F, \neg l')\}]$.

(2.7) **Blocked clause.** A clause C is *blocked* if it contains a literal l such that every clause of F containing $\neg l$ also contains the negation of some other literal of C. Any blocked clause can be removed from F.

Choosing a variable. The main heuristic for choosing a variable is "choose a variable corresponding to the smallest branching number". Although it is possible to figure this out in polynomial time, the heuristic does not look very practical. In fact, in most cases this heuristic can be replaced by something more constructive. The simplest examples are "choose a variable occurring in the

[1] The first preprint appeared in 1994.

shortest clause" and "choose a variable occurring in the largest number of clauses". Typically, an analysis of a splitting algorithm contains a long list of cases corresponding to different syntactic properties of formulas, and the heuristic is to choose a variable certifying at least one of these properties.

Open question. As mentioned above, a typical analysis of a splitting algorithm consists of a long list of cases. The proof corresponding to each case uses a very simple combinatorial argument. It would be interesting to devise a program that, given the desired branching number, generates such a list (i.e., a proof) automatically.

3. PPSZ-like algorithms and derandomization of Satisfiability Coding Lemma

Satisfiability Coding Lemma. On each step DPLL-like algorithms choose a variable for splitting using only "local" properties of the formula. Variables in different branches of the splitting tree are chosen independently. An upper bound for the running time is deduced from recurrent inequalities or from branching numbers for each node of the branching tree.

Paturi, Pudlák and Zane [PPZ97] suggested another ("global") method of bounding the running time of a splitting algorithm. Basically, their algorithm chooses a random permutation of the variables of the input formula and makes splittings in the corresponding order. However, there may be no need to make splitting for *all* variables because the values of some variables in a satisfying assignment can be determined as a result of the use of transformation rules, for example, as a result of unit clause elimination. The analysis of this algorithm is based on the estimate of the number of such variables not requiring splittings. If this estimate is at least s then we can restrict ourselves to splitting trees of depth at most $n - s$. If we find no satisfying assignment after the construction of the tree of depth $n - s$, then the formula is unsatisfiable.

The algorithm of [PPZ97] has several variants, some of them have the best current upper bounds [PPSZ98]. Its simplest version is as follows (for the sake of simplicity we assume $k = 3$):

Algorithm 3.1.

1. Pick a permutation π from S_n at random, where S_n is the set of all permutations of $\{1, \dots, n\}$.
2. Using Procedure 2.1 with the only transformation rule (2.1), construct a splitting tree of depth[2] at most $2n/3$. At each step of splitting, choose the variable x_i such that x_i still occurs in the formula and $\pi(i)$ is the smallest. If a satisfying assignment is found by Procedure 2.1, output it and halt; otherwise output the answer "Unsatisfiable".

Clearly, this algorithm runs in time $\mathrm{poly}(|F|) \cdot 2^{2n/3}$. A proof that this algorithm has admissible probability of error for formulas with at most one satisfying assignment follows from *Satisfiability Coding Lemma* described below.

For the input formula F, we consider a splitting tree constructed in the same way as in Algorithm 3.1 but without the restriction $2n/3$ on the depth. If S is a satisfying assignment for F then the tree has a path leading to S. Each splitting along this path sets a value to a variable. Let S' be the assignment corresponding to all such settings. We emphasize that the values of variables determined by transformation rules are not included in S'. We call S' the *description* of S. By the *length* of the description we mean the number of literals in S'. If we know the description, we can restore the assignment by using transformation rule (2.1).

[2]Clearly, Procedure 2.1 can be easily modified so that it returns `false` if the level of recursion is greater than $2n/3$.

Satisfiability Coding Lemma [PPZ97]. *Let F be a uniquely satisfiable formula in 3-CNF, and S be its satisfying assignment. Consider the description with respect to a permutation chosen uniformly at random from the symmetric group S_n. Then the expected length of the description of S is at most $2n/3$.*

Proof. For each variable x in F, we define an *x-critical clause* to be a clause C such that only the literal corresponding to x is true in C under the satisfying assignment S and all others literals in C are false. Note that for each x, the formula F contains at least one x-critical clause (otherwise, we would get another satisfying assignment by changing the value of x in S). Since we choose the permutation π uniformly at random, the variable x is the last variable (with respect to π) in the x-critical clause C with the probability at least $1/3$. It is easy to see that the last variable of C does not appear in the description of S. Hence, x does not occur in the description with the probability at least $1/3$. The claim now follows from linearity of expectation. □

Let p_ℓ be the probability that the length of the description of the satisfying assignment is exactly ℓ. By Satisfiability Coding Lemma

$$\frac{2n}{3} \geq \sum_{l=0}^{n} p_\ell \ell \geq \left(\left\lfloor \frac{2n}{3} \right\rfloor + 1\right) \sum_{\ell=\lfloor 2n/3 \rfloor+1}^{n} p_\ell.$$

Consequently,

$$\sum_{\ell=\lfloor 2n/3 \rfloor+1}^{n} p_\ell \leq \frac{2n}{3}\left(\left\lfloor \frac{2n}{3} \right\rfloor + 1\right)^{-1} \leq \frac{2n}{3} \cdot \frac{3}{2n+1} = 1 - \frac{1}{2n+1}.$$

Therefore,

$$\sum_{\ell=0}^{\lfloor 2n/3 \rfloor} p_\ell \geq \frac{1}{2n+1},$$

i.e., for a random permutation, the length of a description does not exceed $2n/3$ with the probability at least $1/(2n+1)$. Thus, Algorithm 3.1 has admissible probability of error.

Derandomization of Satisfiability Coding Lemma. How can we get rid of random bits in Algorithm 3.1? One way is to find a "small" set of permutations B_n, for which Satisfiability Coding Lemma still holds. In this case, we could find a required permutation by searching through B_n, not through the set S_n of all permutations. To construct such a set, [PPZ97] uses the space of random 3-wise independent variables. This method gives and "almost" 3-wise independent set of permutations. There are two disadvantages in this approach:

(1) the cardinality of the obtained set of permutations is $O(n^9)$;
(2) the running time of the algorithm increases even more because of "almost" independence instead of "real" independence.

Although these disadvantages do not affect the theoretical bound $\mathrm{poly}(|F|) \cdot 2^{2n/3}$ on the running time of the deterministic algorithm, they make it useless in practice since the huge polynomial factor makes this bound worse than the trivial bound 2^n, at least for $n \leq 200$.

Below we present another (and even simpler) explicit construction of the set B_n of permutations whose cardinality is only $O(n^3)$. Note that the following condition is sufficient for the lemma to hold: for any three distinct positive integers $x, y, z \leq n$ and for a random permutation π from B_n, the probability of $\pi(x) = \max\{\pi(x), \pi(y), \pi(z)\}$ is at least $1/3$. This is a special case of a "max-3-wise independent family of permutations" [BCFM98][3].

The paper [BCFM98] gives no example of such a family of polynomial size; the polynomial-size construction is given only for the case when the uniform distribution on the set of permutations is

[3] In [BCFM98] the symmetric case of min-3-wise permutations is considered.

replaced by non-uniform one. However, this set (and the distribution on it) is obtained as a result of solving a linear program of exponential size. There are approximate constructions in [BCFM98], but they are rather complicated. In addition, due to their approximate nature, the polynomial factor in the running time of the resulting algorithm becomes greater than the corresponding factor for the simple construction presented below. In our construction, B_n will be a multiset; however, it is clear that for the purposes of our algorithm it is sufficient to examine each permutation from B_n only once.

Let p be the smallest prime number such that $n \le p+1$. Clearly, $p \le 2n$. In what follows, $\mathbb{Z}_p$ denotes the finite field with p elements. We consider the projective line P over $\mathbb{Z}_p$, i.e., the classes of elements of $(\mathbb{Z}_p \times \mathbb{Z}_p) \setminus \{(0,0)\}$ with respect to the equivalence relation

$$(x, y) \sim (x', y') \text{ iff } xy' = x'y.$$

Let us consider permutations of its points. The projective line consists of $p+1$ points, so one can identify such permutations with elements of the symmetric group S_{p+1}. We will find a min-3-wise independent subset B_{p+1} of S_{p+1}. The desired multiset B_n of permutations on n symbols will be obtained as a result of restricting[4] permutations from B_{p+1} to the first n symbols. Clearly, the property of being max-3-wise independent still holds for B_n.

As a set B_{p+1} we take the group $PGL(2,p)$ of projective automorphisms of P, i.e., transformations of P, induced by invertible linear transformations of $\mathbb{Z}_p \times \mathbb{Z}_p$: $f((x,y)) = (ax+by, cx+dy)$ with $ad - bc \neq 0$. There are exactly $(p+1)p(p-1)$ different projective automorphisms. It is well known that for any two ordered triples of pairwise distinct points of the projective line, there exists the unique projective automorphism that maps the first triple to the second one (see, e.g., [KM80, Chapter 3, §8, items 8, 9]). In particular, any triple of pairwise distinct points of P is mapped equiprobably onto any other triple under the action of a random permutation from B_{p+1}. Therefore, our construction is correct.

Remark. For any $n \ge k \ge 2$, there exists a multiset $C_{n,k}$ of max-k-wise (or, equivalently, min-k-wise) independent permutations of n symbols such that the cardinality of C does not exceed $n^{(1+1/\log n)k}\mathrm{lcm}(1,\dots,k)$; see [ITT00]. A more careful analysis of the action of the group $PGL(2,p)$ on quadruples of points shows that the multiset B_n constructed above is even a max-4-wise independent family of cardinality $O(n^3)$, see also [Vse00]. In particular, this construction improves the upper bound for the size of the smallest max-4-wise independent family, given in [ITT00].

More complicated algorithms based on Satisfiability Coding Lemma. As shown above, Algorithm 3.1 has admissible probability of error on formulas with at most one satisfying assignment. In fact, this is also true for arbitrary formulas in 3-CNF. Moreover, it is clear that Satisfiability Coding Lemma holds for k-CNF with $k > 3$ (in this case the length of the description is $(1 - 1/k)n$), so an algorithm similar to Algorithm 3.1 can be used to solve k-SAT in $\mathrm{poly}(|F|) \cdot 2^{(1-1/k)n}$ steps.

Derandomization of Algorithm 3.1 for formulas with (possibly) more than one satisfying assignments is however a bit more tricky. Here is the algorithm from [PPZ97]:

Algorithm 3.2.

1. Choose γ such that

$$1 - \gamma/k \approx -\gamma \log_2 \gamma - (1-\gamma)\log_2(1-\gamma).$$

2. For all assignments A such that A sets the values of all n variables and contains at most γn positive literals, check whether A satisfies F and if so, output it and halt.

[4] By the restriction of a permutation π on the first n symbols we mean the unique permutation $\pi' \in S_n$ such that for all $x, y \in [1..n]$ the inequality $\pi'(x) < \pi'(y)$ holds if and only if $\pi(x) < \pi(y)$. Namely, $\pi'(k)$ is the cardinality of the set $\{\pi(1), \dots, \pi(n)\} \cap [1..\pi(k)]$.

3. For all permutations $\pi \in B_n$, construct a splitting tree of depth at most $n(1-\gamma/k)+1$ using Procedure 2.1 with the only transformation rule (2.1). At each step of splitting, choose a variable x_i such that x_i still occurs in the formula and $\pi(i)$ is the smallest possible. If a satisfying assignment is found by Procedure 2.1, output it and halt.
4. Answer "Unsatisfiable".

Algorithm 3.2 has a worse running time bound than Algorithm 3.1; for formulas in k-CNF, its running time is $\mathrm{poly}(|F|) \cdot 2^{n(1-\gamma/k)}$ which becomes $\mathrm{poly}(|F|) \cdot 2^{0.896n}$ for 3-CNF.

Satisfiability Coding Lemma counts only the variables that are omitted from the description of a satisfying assignment because of the application of rule (2.1) to a variable x in the following case:

- there is an x-critical clause consisting of x, y and z,
- $\pi(x) > \pi(y)$, and
- $\pi(x) > \pi(z)$.

However, the condition $\pi(x) > \pi(z)$ is not necessary for a variable z to disappear from x-critical clause. For example, z itself can be eliminated because of the application of rule (2.1). A nice way to use and count such dependencies is described in [PPSZ98]:

Algorithm 3.3.

1. Add to F all clauses that can be derived from it by resolution bounded to the clauses of size at most $r(n)$.
2. Perform as in Algorithm 3.1, but for smaller depth $d(n)$ of the tree.

The analysis of this algorithm and its derandomized version is rather complicated; the results of [PPSZ98] are $\mathrm{poly}(|F|) \cdot 2^{0.448n}$ for 3-SAT and $\mathrm{poly}(|F|) \cdot 2^{0.387n}$ for uniquely satisfiable formulas in 3-CNF.

In general, the framework created by Algorithm 3.1 can be described in the following way:

Algorithm 3.4.

1. Construct a splitting tree of *certain depth* using Procedure 2.1 with *certain transformation rules*. For each splitting, choose a variable uniformly at random from the variables still occurring in the formula. If a satisfying assignment is found by Procedure 2.1, then output it and halt; otherwise output the answer "Unsatisfiable".

Open problems. This section and the paper [Vse00] construct the sets B_n for $k = 3$ and for $k = 4$ respectively. These constructions, which are of independent interest, are based on the properties of the groups $PGL(k, p^s)$ for $k = 3, 4$. Is it possible to generalize these constructions for an arbitrary k? It is conjectured in [Vse00] that the group $PGL(k-1, p^s)$ acting on the points of $(k-2)$-dimensional projective space is min-k-wise (max-k-wise) independent for some linear order on the points of the projective space.

4. Two more proofs of the Valiant-Vazirani Lemma

The Valiant-Vazirani Lemma [VV86] is one of the first non-trivial results on the relations between complexity classes.

The Valiant-Vazirani Lemma. *There exists a randomized (with one-sided error) polynomial-time reduction of the satisfiability problem to its instances with at most one satisfying assignment.*

In other words, given a formula F in CNF, one can construct formulas $F_1, \dots, F_m$ in CNF such that

- if F is satisfiable then, with probability greater than $1/2$, at least one formula of $F_1, \dots, F_m$ has exactly one satisfying assignment;
- if F is unsatisfiable, then all formulas $F_1, \dots, F_m$ are unsatisfiable.

A generalization of the Valiant-Vazirani Lemma is used, for example, in the famous theorem of Toda [Tod91], which claims that any language of the polynomial hierarchy is Turing reducible (in deterministic polynomial time) to a language from the class $\mathcal{PP}$. A similar statement would be also useful in our context, since current upper bounds for formulas in 3-CNF with a unique satisfying assignment [PPSZ98] (see section 3) are better than the bounds for arbitrary formulas [Sch99] (see section 5).

There are many different proofs of the lemma [BF97, Cha94, CRS93, MVV87]. Also, there are several proofs of a close result: the existence of a reduction to formulas with odd (or zero) number of satisfying assignments [Gup93, NRS95]. In this section we give two new proofs. Both our proofs are based on the idea used in [BF97]. In [BF97] this idea is combined with the use of the Kolmogorov complexity. In our proofs, we get rid of the application of the Kolmogorov complexity. This simplifies the proof in [BF97] and displays its number-theoretic essence.

Remark. As we mentioned above, a statement similar to the Valiant-Vazirani Lemma would be useful in the context of our paper. However, for this purpose we need a reduction satisfying an additional requirement: given a formula in 3-CNF, the reduction is to output formula(s) in 3-CNF with the same (or almost the same) number of variables. The original reduction of Valiant and Vazirani adds $\Omega(n)$ equalities of the form $l_1 \oplus l_2 \oplus \dots \oplus l_n = 0$ to the initial formula, where $l_i = x_i$ or $l_i = \neg x_i$ (decided randomly), and $\oplus$ denotes the addition modulo 2. To represent all these equalities in 3-CNF, one has to introduce $\Omega(n^2)$ new variables. Furthermore, none of other known reductions (including ours) satisfies the above requirement.

The first reduction. Let F be a formula and A be an assignment to all its variables. We identify A with an n-bit number $a = a_0 a_1 \dots a_{n-1}$ such that $a_j = 1$ if the corresponding variable in A has the value `true`, and $a_j = 0$, otherwise. We choose integers p_i and r_i as follows. First, we choose $i \in [0..n]$ uniformly at random. Second, we choose $p_i \in [1..b_i]$ and $r_i \in [0..b_i]$ uniformly at random, where $b_i = 4 \cdot 2^i n^2$. Then we replace F by the formula

$$F \wedge (a \bmod p_i = r_i).$$

Here "$(a \bmod p_i = r_i)$" stands for a propositional formula in CNF in the variables $a_0, \dots, a_{n-1}$ (possibly using also some auxiliary variables), which represents the corresponding arithmetic congruence. For example, this formula can be obtained by encoding the standard column multiplication. Obviously, this reduction takes polynomial time and transforms an unsatisfiable formula into an unsatisfiable formula. It remains to prove that if F is satisfiable then, with high probability, the new formula is uniquely satisfiable.

Let $a^{(1)}, \dots, a^{(D)}$ be all satisfying assignments of the formula F. Note that $i = \lceil \log_2 D \rceil$ with probability $1/(n+1)$. Suppose that this event happened. Note that for given j and h ($j \neq h$) there are at most n prime divisors of the difference $a^{(j)} - a^{(h)}$. On the other hand, for sufficiently large n, there are at least $0.92129 \cdot b_i / \ln b_i > b_i / \log_2 b_i \geq 2^{i+1} n$ primes not greater than b_i [Че655]. Thus, there are at least $2^{i+1}n - 2^i n = 2^i n$ numbers p not exceeding b_i such that the remainder of the satisfying assignment $a^{(j)}$ modulo p differs from the remainders of all other satisfying assignments modulo p. Therefore, at least $2^i n$ such pairs $0 \le p_i, r_i \le b_i$ "distinguish" the assignment $a^{(j)}$ from all other satisfying assignments. Note that for different assignments the sets of distinguishing pairs are disjoint. Hence, there are at least $2^i n \cdot D \ge 2^{2i-1} n$ required pairs (p_i, r_i). Thus, for sufficiently large n the probability to choose such a pair is not less than $\frac{2^{2i-1}n}{16 \cdot 2^{2i} n^4} = \frac{1}{32n^3}$.

Multiplying by the probability to choose the "correct" i, we have that the probability of error of our reduction is at most $1 - \frac{1}{32n^4+32}$. Choosing triples (i, p_i, r_i) at random $O(n^4)$ times, we get a constant error probability.

The second reduction. Let $\mathbb{Z}_p[t]$ denote the ring of polynomials in one variable over the finite field with p elements. It is well known that the ring $\mathbb{Z}_p[t]$ is the "right" analogue of the ring $\mathbb{Z}$ from the arithmetical point of view. On the other hand, unlike asymptotic formulas for the number of primes in an interval, there is an explicit exact formula (which has an elementary proof) for the number of irreducible polynomials of a given degree. In addition, the propositional formula $(a \bmod p_i = r_i)$, where a, p_i, r_i are the polynomials over $\mathbb{Z}_p$, is simpler since there are no shifts from one digit to another. This might be useful in view of Open Question 2 (see below). Also note that an approach similar to what follows can be developed for an arbitrary finite field, not only for $\mathbb{Z}_2$.

In the first reduction, we identified the assignment A with an n-bit number $a = a_0, \ldots, a_{n-1}$. Now we identify A with the coefficients of the polynomial $a(t) = a_0 + a_1 t + \cdots + a_{n-1} t^{n-1}$ over the field $\mathbb{Z}_2$, where a_j are the same as in the first reduction. Again, we choose $i \in [0..n]$ at random and replace the input formula F by the formula

$$F \wedge (a \bmod p_i = r_i),$$

where p_i, r_i are now randomly chosen polynomials such that $\deg p_i = d_i$ (i.e., $p_i = x_i^{d_i} + \sum_{t=0}^{d_i-1} c_t x_i^t$, where c_t are chosen at random), $\deg r_i < d_i$ and $d_i = i + \lceil \log_2 n \rceil + 4$.

Suppose that $i = \lceil \log_2 D \rceil$. Take $d = d_i$. Denote by N_d the number of irreducible polynomials over $\mathbb{Z}_2$ of degree d. There is an explicit formula for N_d (see, e.g., [IR82, Chapter 7, §2, Corollary 2]). Namely,

$$N_d = \frac{\sum_{s|d} \mu(\frac{d}{s}) 2^s}{d},$$

where μ denotes the Möbius function:

- $\mu(1) = 1$;
- $\mu(h) = 0$, if h is not squarefree;
- $\mu(h) = (-1)^u$, if $h = q_1 \ldots q_u$, and $q_1, \ldots, q_u$ are pairwise distinct primes.

Since $\mu(h) \ge -1$ holds for all $h > 1$ and $\mu(1) = 1$, we have

$$N_d \ge \frac{2^d - \sum_{s \le d/2} 2^s}{d} \ge \frac{2^d - 2^{d/2+1}}{d} \ge \frac{2^{d-1}}{d}.$$

On the other hand, each of the $D-1$ differences $a^{(j)} - a^{(h)}$ (with $j \ne h$) has at most n/d irreducible factors of degree d. Thus, at least $N_d - \frac{nD}{d} \ge \frac{2^{d-1}}{d} - \frac{2^{d-4}}{d} \ge \frac{2^{d-2}}{d}$ pairs (p_i, r_i) with $\deg p_i = d$, $\deg r_i < d$ "distinguish" the assignment $a^{(j)}$ from all other satisfying assignments.

Similarly to the first reduction, there are at least $\frac{2^{d-2}}{d} \cdot D$ required pairs. The probability to choose such a pair is not less than

$$\frac{2^{d-2} D}{d 2^{2d-1}} \ge \frac{1}{64nd} \ge \frac{1}{128n^2}$$

for $n \ge 8$. Multiplying by the probability to choose the correct i, we have that the probability of error of our reduction is at most $1 - \frac{1}{128n^3+128}$, i.e., by repeating the random choices $O(n^3)$ times we can get a constant error probability.

Open questions.

1. The question of the derandomization of the Valiant-Vazirani Lemma is open. For example, is it possible to replace a randomized polynomial-time reduction by a deterministic reduction running in time $\mathrm{poly}(|F|) \cdot c^n$ for some constant $c < 2$? The similar question for a possibly weaker reduction of satisfiability to its instances with zero or odd number of satisfying assignments is also open.
2. All known proofs of the Valiant-Vazirani Lemma reduce satisfiability to formulas in CNF with arbitrary long clauses, even if the input formula is in 3-CNF. A further reduction to 3-CNF can increase the number of variables in the formula significantly. Is there a natural reduction to formulas in 3-CNF such that the increase in the number of variables is not too large, for example, only $o(n)$ new variables appear?

5. Local search algorithms

Local search algorithms for SAT include greedy search [KP92, SLM92], "cautious" search [GW93], random walk [Pap91, Sch99], other strategies and their combinations (see [GPFW00] for survey). Although many of these algorithms are well studied experimentally, good upper bounds are proven only for *random walk algorithms*, the simplest of them. In this section we discuss random walk algorithms for SAT, their derandomization, and the possibility to combine the random walk approach and the splitting approach (viewed as backtracking) in one algorithm.

Random walk algorithms. Random walk algorithms for SAT are very simple randomized algorithms that start from an initial assignment chosen at random and move to a satisfying assignment step by step: at each step, the algorithm flips the value of a variable chosen at random from an unsatisfied clause. Such algorithms solve 2-SAT in polynomial time [Pap91] and solve k-SAT in time $(2-2/k)^n$ up to a polynomial factor, where n is the number of variables in the input formula [Sch99]. In particular, for $k = 3$ this bound is $(4/3)^n$, currently the best known upper bound for 3-SAT algorithms.

The following algorithm (parameterized by two functions α and τ) represents a family of random walk algorithms for SAT. Given an input formula F with n variables, the algorithm performs at most $\alpha(n)$ walks starting from random initial assignments; each walk consists of at most $\tau(n)$ steps.

Algorithm 5.1.

1. Repeat $\alpha(n)$ times:
   1a. Choose an assignment A uniformly at random.
   1b. If A satisfies F, return A and halt. Otherwise repeat the following instructions $\tau(n)$ times:
   - Take any unsatisfied clause C in F.
   - Choose a variable x uniformly at random from the variables occurring in C.
   - Modify A by flipping the value of x in A.
   - If the updated assignment A satisfies F, return A and halt.
2. Return "Unsatisfiable" and halt.

Algorithm 5.1 with $\alpha(n) = 1$ and $\tau(n) = 2n^2$ is Papadimitriou's polynomial-time algorithm for 2-SAT [Pap91]. Algorithm 5.1 with $\alpha(n) = (2-2/k)^n$ and $\tau(n) = 3n$ is Schöning's $(2-2/k)^n$-time algorithm for k-SAT [Sch99].

To analyse Algorithm 5.1, we use its connection with *one-dimensional random walks* (see, e.g., [Fel68]). Consider the performance of this algorithm on a formula in k-CNF. Suppose that the input formula has a satisfying assignment S that differs from an initial assignment A in the values of exactly i variables. Note that at each step, the algorithm moves closer to S with probability at least $1/k$ because an unsatisfied clause always contains at least one variable whose values in S

and in the current assignment are different. Thus, the modification process that starts from A is related to the following one-dimensional random walk.

Consider a particle walking on the interval $[0..n]$. The particle starts from the position i at time $t = 0$ and walks for $\tau(n)$ steps. At each step, if the particle's position is j where $0 < j < n$, it moves to $j-1$ with probability $1/k$ and moves to $j+1$ with probability $1 - 1/k$. If $j = 0$, the particle remains at the same position with probability 1. If $j = n$, the particle moves to $j-1$ with probability 1. We denote by $p_{i,t}$ the probability that the particle starting from i reaches 0 in t steps. The following lemma expresses the connection between $p_{i,t}$ and the error probability of Algorithm 5.1.

Lemma 5.1. *The error probability of Algorithm 5.1 is not greater than*

$$\exp\left(-\frac{\alpha(n)}{2^n}\sum_{i=0}^{n}\binom{n}{i}p_{i,\tau(n)}\right).$$

Proof. It suffices to consider the case of a satisfiable input formula. Let S be any satisfying assignment. Consider one of the $\alpha(n)$ walks performed by the algorithm. For any i, an initial assignment A differs from S in the values of exactly i variables with probability $\binom{n}{i}/2^n$. It is easy to see that such a walk finds S (or another satisfying assignment if some preceding assignment along the walk happens to be satisfiable) with probability at least $p_{i,\tau(n)}$. Summing up for all possible i's, we get the bound

$$p = \sum_{i=0}^{n}\frac{\binom{n}{i}}{2^n}p_{i,\tau(n)}$$

on the probability that a walk finds a satisfying assignment in $\tau(n)$ steps. The error probability of Algorithm 5.1 is therefore not greater than

$$\begin{aligned}(1-p)^{\alpha(n)} &= \exp\left(\alpha(n)\ln(1-p)\right) \le \\ &\le \exp(-\alpha(n)\cdot p) = \exp\left(-\alpha(n)\sum_{i=0}^{n}\frac{\binom{n}{i}}{2^n}p_{i,\tau(n)}\right). \qquad \square\end{aligned}$$

Papadimitriou's algorithm. Papadimitriou [Pap91] proves that Algorithm 5.1 with $\alpha(n) = 1$ and $\tau(n) = 2n^2$ (i.e., running in polynomial time) solves 2-SAT with admissible probability of error. The proof is based on the following observation: when we flip the value of a variable chosen at random from an unsatisfied 2-clause, we move closer to a satisfying assignment with probability at least 1/2. The probability $p_{i,2n^2}$ for the corresponding one-dimensional random walk is at least 1/2 [Pap91, Pap94]; hence the error probability of this algorithm is admissible.

We show that the performance of Papadimitriou's algorithm on renamable Horn formulas without unit clauses can be described by the same one-dimensional random walk. Thus, this algorithm (extended by unit clause elimination) can be used for computing a satisfying assignment for a renamable Horn formula in polynomial time with admissible probability of error.

Recall that a formula F is called a *Horn* formula if each clause in F contains at most one positive literal. Let l be a literal occurring in a formula F. By *reversing* l in F we mean the replacement of all occurrences of l and $\neg l$ by $\neg l$ and l respectively. A *renamable Horn* formula is defined to be a formula that can be transformed into a Horn formula by reversing some literals [Lew78]. There is a number of polynomial-time algorithms that solve SAT for renamable Horn formulas, for example SLUR (Single Lookahead Unit Resolution) [FV98, SAFS95].

Note that if a renamable Horn formula F contains no unit clauses then F is satisfiable. This observation leads to the following algorithm. First, we eliminate all unit clauses (if any) from the input formula using transformation rule (2.1). It is easy to see that unit clause elimination transforms a renamable Horn formula into a renamable Horn one. Then we apply Papadimitriou's algorithm to find a satisfying assignment.

Theorem 5.2. *Let F be a renamable Horn formula without unit clauses. Algorithm 5.1 with $\alpha(n) = 1$ and $\tau(n) = 2n^2$ finds a satisfying assignment for F in polynomial time with admissible probability of error.*

Proof. If suffices to prove that the performance of Algorithm 5.1 on F has the same feature as its performance on formulas in 2-CNF: when we flip the value of a variable chosen at random from an unsatisfied clause, we move closer to a satisfying assignment with probability at least $1/2$.

Clearly, if F is a Horn formula without unit clauses, then F is satisfied by the assignment S_0 in which all variables have the value `false`. Furthemore, for each clause C in F the following holds: all literals of C except at most one are true under S_0. It is easy to see that the case of renamable Horn formulas is similar to the case of Horn ones. Namely, if F is a renamable Horn formula without unit clauses, then F has a satisfying assignment S such that, for each clause C, all literals of C except at most one are true under S.

Consider the choice of a variable for flipping from an unsatisfied clause C. Let k be the number of literals in C. All these literals are false under the current assignment. At the same time, all of them except at most one must be true under S. Therefore, when we flip the value of one of these k literals, we move closer to S with probability at least $\frac{k-1}{k}$. Since F contains no unit clause, this probability is not less than $1/2$. □

Schöning's algorithm. Schöning [Sch99] proves that k-SAT can be solved by Algorithm 5.1 with $\alpha(n) = (2 - 2/k)^n$ and $\tau(n) = 3n$ with admissible probability of error. The proof is based on the following estimation of the probability $p_{i,3n}$: it is shown that $p_{i,3n} \geq (k-1)^{-i}$ up to a polynomial factor for any $k \geq 3$ and any i. Then Lemma 5.1 gives the upper bound

$$\exp\left(-\frac{(2-2/k)^n}{2^n}\sum_{i=0}^{n}\binom{n}{i}\frac{(k-1)^{-i}}{\mathrm{poly}(n)}\right) =$$
$$= \exp\left(\frac{(1-1/k)^n}{\mathrm{poly}(n)}\left(1+\frac{1}{k-1}\right)^n\right) = \exp\left(-\frac{1}{\mathrm{poly}(n)}\right) = 1 - \frac{1}{\mathrm{poly}(n)}$$

for the probability of error for Schöning's algorithm. The last equality[5] follows from $\exp(-x) \leq 1 - x/e$ which holds for any $x < 1$.

Derandomization of Schöning's algorithm. A deterministic version of Schöning's algorithm is described in [DGH+00, DGHS00]. The idea behind the derandomization is simple and intuitive. First, we cover the whole search space (all 2^n possible assignments) by Hamming balls of some fixed radius R. This covering should be minimal, i.e., we try to use as few balls as possible. Then we take every ball and search for a satisfying assignment inside it. It is clear that there is trade-off between the number of balls and the time of searching inside a ball: the more balls we use, the faster search inside a ball is. Thus, the number of balls (or, equivalently, the radius R) has the "optimal" value which minimizes the upper bound on the overall running time. As shown in [DGH+00, DGHS00], when the input formula is in k-CNF, the "optimal" value of R is $\frac{n}{k+1}$. The overall running time of the resulting algorithm is $(2 - \frac{2}{k+1} + \epsilon)^n$ up to a polynomial factor, which is the best known upper bound for deterministic k-SAT algorithms.

To find a satisfying assignment inside a ball, the algorithm [DGH+00, DGHS00] uses a very simple procedure based on local search with backtracking. Namely, take an unsatisfied clause $l_1 \vee \ldots \vee l_t$ and consider all the t possibilities: the value of l_1 is wrong, ... , the value of l_t is wrong. The depth of the tree constructed in this way can be limited to R, and the number of nodes is thus $O(k^R)$. Using a more efficient (but more complicated) version of this procedure, one can improve the bound on the size of this tree and therefore on the overall running time of the algorithm. For 3-SAT, the improved algorithm runs in time $\mathrm{poly}(|F|) \cdot 1.481^n$ [DGH+00, DGHS00].

It is also possible to get rid of the "$+\epsilon$" summand in the running time of this algorithm at the cost of using exponential space [DGH+00].

[5]See Sect. 1 for our convention about using the poly notation.

Random walk algorithms with the "back button". Is it possible to incorporate *backtracking* into random walk algorithms for SAT? We can do it using the recent approach of Fagin et al. [FKK+00] who introduced *Markov chains with the "back button"* (the study of such processes is motivated by connection with browsing on the world-wide web). Namely, we allow each step to be, with some probability, a *backward step*. Such a step, like a click on the "back button", undoes the last flip in the current assignment. Thus backward steps can be viewed as a kind of backtracking for random walk algorithms.

The purpose of this section is to understand whether the "back button" can change the probability to find a satisfying assignment. To answer this question, we compare two types of random walks on the line, with and without backward steps. Using results from [FKK+00], we give comparative bounds on the success probabilities of polynomially long walks. Namely, we show that backward steps cannot increase the success probability by more than a polynomial factor, i.e., the "back button" does not give a substantial gain. We also show that if the probability of backward steps is not greater than 0.5 then the "back button" does not lead to a substantial loss.

We modify Algorithm 5.1 (denoted by RW – Random Walks) into Algorithm 5.2 (denoted by RWB – Random Walks with Back button). The new algorithm RWB keeps a *history stack* H whose elements are assignments. The top element of the stack corresponds to the current assignment. Like RW, the algorithm RWB performs at most $\alpha(n)$ walks; each walk consists of at most $\tau(n)$ steps but each step is either a *forward step* or a *backward step*. In a forward step, RWB modifies the current assignment in the same way as RW; if the updated assignment A satisfies F then RWB returns A and halts; otherwise RWB pushes the updated A onto H. A backward step is that RWB pops the top element from H (the new top element thereby becomes the current assignment). Starting with the empty history stack, RWB performs as follows:

Algorithm 5.2.

1. Repeat $\alpha(n)$ times:
   (1) 1a. Choose an assignment A uniformly at random.
   (2) 1b. If A satisfies F, return A and halt. Otherwise push A onto H and repeat the following instruction $\tau(n)$ times:
      - If H contains only one element, make a forward step. Otherwise make a backward step with probability b and a forward step with probability $1 - b$.
2. Return "Unsatisfiable" and halt.

The probability b is called the *backoff probability*.

To represent the performance of RWB using one-dimensional random walks we modify the one-dimensional random walk model for RW (described after Algorithm 5.1 above) as follows. The walk for RWB keeps a history stack H whose elements are the particle's positions. Each step of the walk is either a forward step or a backward step. In a forward step, the particle moves according to the rules for RW and its new position is pushed onto H. In a backward step, the top element is popped from H and the particle moves to the position that is the top element of the updated stack. At time $t = 0$ the history H contains only the initial position i. Then the particle takes steps similar to RWB: if H contains only one element, the particle takes a forward step; otherwise takes a backward step with probability b and a forward step with probability $1 - b$. We denote by $p_{i,t}^{(b)}$ the probability for the particle to reach 0 in at most t steps.

Lemma 5.3. *The error probability of Algorithm 5.2 is not greater than*

$$\exp\left(-\frac{\alpha(n)}{2^n}\sum_{i=0}^{n}\binom{n}{i}p_{i,\tau(n)}^{(b)}\right).$$

Proof. Similar to the proof of Lemma 5.1. □

Lemma 5.1 and Lemma 5.3 show that lower bounds on $p_{i,t}$ and $p_{i,t}^{(b)}$ imply upper bounds on the error probabilities of RW and RWB respectively. Thus, comparative bounds on $p_{i,t}$ and $p_{i,t}^{(b)}$ allow us to compare the upper bounds for RW and RWB (obtained using Lemmas 5.1 and 5.3).

We now prove two theorems. The first theorem shows that the "back button" cannot increase the probability of reaching 0 by more than a polynomial factor. The second one shows that the "back button" (with $b \le 0.5$) cannot lower this probability by more than a polynomial factor if we admit a quadratic increase in the walk length. Both theorems are proven using results from [FKK+00].

Let $X_{i,t}$ and $X_{i,t}^{(b)}$ denote the particle's position at time t in the random walks starting from i for RW and RWB respectively. Let H_t denote the history stack of the random walk for RWB at time t; following [FKK+00], the length $l(H_t)$ of the history stack is defined as the number of particle positions stored in it *minus* 1 (i.e., we do not count the initial position).

Theorem 5.4. *For any backoff probability b,*

$$p_{i,t}^{(b)} \;\le\; (t+1)^2 \cdot p_{i,t}.$$

Proof. Note that $p_{i,t}^{(b)}$ is upper-bounded by the sum of $t+1$ probabilities $Pr\{X_{i,t'}^{(b)} = 0\}$ for $t' = 0, 1, \ldots, t$. Let $t' \in [0..t]$ be the value maximizing $Pr\{X_{i,t'}^{(b)} = 0\}$. Since $0 \le l(H_s) \le s$ for any s, we have

$$\begin{aligned} p_{i,t}^{(b)} &\le (t+1) Pr\{X_{i,t'}^{(b)} = 0\} = \\ &= (t+1) \sum_{\lambda=0}^{t'} Pr\{X_{i,t'}^{(b)} = 0 \mid l(H_{t'}) = \lambda\} \cdot Pr\{l(H_{t'}) = \lambda\} \le \\ &\le (t+1)(t'+1) \cdot \max_{0 \le \lambda \le t'} Pr\{X_{i,t'}^{(b)} = 0 \mid l(H_{t'}) = \lambda\}. \end{aligned}$$

By Theorem 3.1 from [FKK+00],

$$Pr\{X_{i,t'}^{(b)} = 0 \mid l(H_{t'}) = \lambda\} = p_{i,\lambda}.$$

Since $p_{i,\lambda} \le p_{i,t}$ for $0 \le \lambda \le t' \le t$, the claim follows. □

Theorem 5.5. *For any backoff probability $b \le 0.5$,*

$$p_{i,t^2}^{(b)} \;\ge\; p_{i,t} \,/\, poly(t).$$

Proof. Again, using Theorem 3.1 from [FKK+00], we have

$$\begin{aligned} p_{i,t^2}^{(b)} &\ge Pr\{X_{i,t^2}^{(b)} = 0\} = \\ &= \sum_{\lambda=0}^{t^2} Pr\{X_{i,t^2}^{(b)} = 0 \mid l(H_{t^2}) = \lambda\} \cdot Pr\{l(H_{t^2}) = \lambda\} = \\ &= \sum_{\lambda=0}^{t^2} p_{i,\lambda} \cdot Pr\{l(H_{t^2}) = \lambda\}. \end{aligned}$$

To estimate $Pr\{l(H_{t^2}) = \lambda\}$ for $b \le 0.5$, we first estimate $Pr\{l(H_{t^2}) = t\}$ for $b = 0.5$. Consider the following random walk on the line: a particle starts from 0 and moves left and right with probabilities equal to 0.5. It is easy to see that $Pr\{l(H_s) = \mu\}$ for $b = 0.5$ is exactly the probability that the particle's position at time s is either μ or $-\mu$. In particular, $Pr\{l(H_{t^2}) = t\} = 2r_t$, where

r_t denotes the probability that the particle's position at time t^2 is t. The probability r_t is exactly $\binom{t^2}{(t^2-t)/2} \cdot 2^{-t^2}$ which is within a polynomial of

$$\frac{t^{2t^2}}{2^{t^2} \cdot (\frac{t^2-t}{2})^{(t^2-t)/2} \cdot (\frac{t^2+t}{2})^{(t^2+t)/2}}$$

(by Stirling's formula). Straightforward calculation shows that for sufficiently large t the last expression is greater than a constant. Thus, the claim for $b = 0.5$ follows.

If $b \le 0.5$, the probability $Pr\{l(H_{t^2}) \ge t\}$ is even greater than it is for $b = 0.5$ (and obviously greater than the probability $Pr\{l(H_{t^2}) = t\}$ estimated above). In particular, this probability is greater than $1/\text{poly}(t)$. Recall that $Pr\{l(H_{t^2}) = \lambda\} = 0$ for $\lambda > t^2$. Hence, there is a λ such that $t \le \lambda \le t^2$ and $Pr\{l(H_{t^2}) = \lambda\} \ge Pr\{l(H_{t^2}) \ge t\}/t^2 \ge 1/\text{poly}(t)$. Since $p_{i,\lambda} \ge p_{i,t}$ when $\lambda \ge t$, the claim follows. $\square$

Open Problems.

1. Note that while the existence of critical clauses improves the performance of PPSZ-like algorithms, it is the real bottleneck in random walk algorithms: if the unsatisfied clause we choose is not critical, then the probability of going in the "right" direction is at least $2/k$ and not $1/k$. It would be interesting to make use of this trade-off by combining the two approaches in one algorithm.
2. We have shown that RWB is similar to RW when walks are polynomially long. The question of whether the similarity holds for exponentially long walks is open.

References

[AL97] S. Arora, C. Lund, *Hardness of approximation*, Approximation algorithms for NP-hard problems (D. Hochbaum, ed.), PWS Publishing Company, Boston, 1997.

[Bei00] R. Beigel, *Finding maximum independent sets in sparse and general graphs*, Proceedings of SODA'99, 1999, pp. 856–857.

[BE95] R. Beigel, D. Eppstein, *3-coloring in time $O(1.3446^n)$: a no-MIS algorithm*, Proceedings of FOCS'95, 1995, pp. 444–452.

[BCFM98] A. Broder, M. Charikar, A. Frieze, M. Mitzenmacher, *Min-wise independent permutations*, Proceedings of STOC'98, 1998, pp. 327–336.

[BF97] H. Buhrman, L. Fortnow, *Resource-bounded Kolmogorov complexity revisited*, Proceedings of STACS'97, LNCS **1200**, Springer-Verlag, pp. 105–116.

[Cha94] S. Chari, *Randomness as a computational resource: issues in efficient computation*, PhD thesis, 1994.

[CRS93] S. Chari, P. Rohatgi, A. Srinivasan, *Improved algorithms via approximations of probability distributions*, Journal of Computer and System Sciences **61** (2000), 81–107.

[Чеб55] П. Л. Чебышев, *О простых числах*, в кн.: П. Л. Чебышев, *Избранные труды*, М., Изд-во АН СССР, 1955, pp. 33–54, (P. L. Chebyshev, *On prime numbers*).

[Дан81] Е. Я. Данцин, *Две системы доказательства тавтологичности, основанные на методе расщеплений*, Записки научных семинаров ЛОМИ **105** (1981), 24–44; English translation: E. Ya. Dantsin, *Two propositional proof systems based on the splitting method*, Journal of Soviet Mathematics, 22(3):1293–1305, 1983.

[DGHK01] E. Dantsin, M. Gavrilovich, E. A. Hirsch, B. Konev, *Approximation algorithms for MAX SAT: a better performance ratio at the cost of a longer running time*, Annals of Pure and Applied Logic (2001) (to appear).

[DGH+00] 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*, submitted (2000).

[DGHS00] E. Dantsin, A. Goerdt, E. A. Hirsch, U. Schöning, *Deterministic algorithms for k-SAT based on covering codes and local search*, Proceedings of ICALP'2000 (U. Montanari, J. D. P. Rolim, E. Welzl, eds.), LNCS **1853**, Springer-Verlag, 2000.

[DLL62] M. Davis, G. Logemann, D. Loveland, *A machine program for theorem-proving*, Communications of the ACM **5(7)** (1962), 394–397.

[DP60] M. Davis, H. Putnam, *A computing procedure for quantification theory*, Journal of the ACM **7(3)** (1960), 201–215.

[FKK+00] R. Fagin, A. Karlin, J. Kleinberg, P. Raghavan, S. Rajagopalan, R. Rubinfeld, M. Sudan, A. Tomkins, *Random walks with "back buttons"*, Proceedings of STOC'2000, 2000, pp. 484–493.

[Fel68] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1, Wiley, 1968.

[FV98] J. Franco, A. Van Gelder, *A perspective on certain polynomial time solvable classes of satisfiability*, submitted to *Discrete Applied Mathematics* (1998).

[GW93] I. P. Gent, T. Walsh, *Towards an understanding of hill-climbing procedures for SAT*, Proceedings of AAAI-93, AAAI Press / MIT Press, 1993, pp. 28–33.

[GHNR00] J. Gramm, E. A. Hirsch, R. Niedermeier, P. Rossmanith, *New worst-case upper bounds for MAX-2-SAT with application to MAX-CUT*, Electronic Colloquium on Computational Complexity, Technical Report **00-037**, 2000, ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/2000/TR00-037/index.html.

[GPFW00] J. Gu, P. Purdom, J. Franco, B. W. Wah, *Algorithms for the satisfiability problem*, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2000.

[Gup93] S. Gupta, *On bounded-probability operators and $C_{=}P$*, Information Processing Letters **48** (1993), 93–98.

[Hir00a] E. A. Hirsch, *New worst-case upper bounds for SAT*, Journal of Automated Reasoning **24(4)** (2000), 397–420.

[Hir00b] E. A. Hirsch, *Worst-case time bounds for MAX-k-SAT w.r.t. the number of variables using local search*, Proceedings of RANDOM 2000, ICALP Workshops 2000 (J. D. P. Rolim et al., eds.), Proceedings in Informatics **8**, Carleton Scientific, 2000, pp. 69–76.

[ITT00] T. Itoh, Y. Takei, J. Tarui, *On permutations with limited independence*, Proceedings of SODA 2000, 2000, pp. 137–146.

[IR82] K. Ireland, M. Rosen, *A Classical Introduction to Modern Number Theory*, Graduate Texts in Mathematics, **87**, Springer-Verlag, 1982.

[KM80] А. И. Кострикин, Ю. И. Манин, *Линейная алгебра и геометрия*, М., Изд-во МГУ, 1980; English translation: A. Kostrikin, Yu. Manin, *Linear Algebra and Geometry*, Gordon & Breach, 1989.

[KP92] E. Koutsoupias, C. H. Papadimitriou, *On the greedy algorithm for satisfiability*, Information Processing Letters **43(1)** (1992), 53–55.

[Kul98] O. Kullmann, *Heuristics for SAT algorithms: Searching for some foundations*, Discrete Applied Mathematics (submitted) (1998).

[Kul99] O. Kullmann, *New methods for 3-SAT decision and worst-case analysis*, Theoretical Computer Science **223 (1-2)** (1999), 1–72.

[Kul00] O. Kullmann, *Investigations on autark assignments*, Discrete Applied Mathematics (2000) (to appear).

[KL97] O. Kullmann and H. Luckhardt, *Deciding propositional tautologies: Algorithms and their complexity* (1997), preprint, available from http://www.cs.toronto.edu/~kullmann.

[Lew78] H. R. Lewis, *Renaming a set of clauses as a Horn set*, Journal of the ACM **25(1)** (1978), 134–135.

[MS85] B. Monien, E. Speckenmeyer, *Solving satisfiability in less then 2^n steps*, Discrete Applied Mathematics **10** (1985), 287–295.

[MVV87] K. Mulmuley, U. V. Vazirani, V. V. Vazirani, *Matching is as easy as matrix inversion*, Combinatorica **7(1)** (1987), 105–113.

[NR00] R. Niedermeier, P. Rossmanith, *New upper bounds for MaxSat*, Journal of Algorithms (2000) (to appear).

[NRS95] A. V. Naik, K. W. Regan, D. Sivakumar, *Quasilinear Time Complexity Theory*, Theoretical Computer Science **148** (1995), 325-349.

[PPSZ98] R. Paturi, P. Pudlák, M. E. Saks, F. Zane, *An improved exponential-time algorithm for k-SAT*, Proceedings of FOCS'98, 1998, pp. 628–637.

[PPZ97] R. Paturi, P. Pudlák, F. Zane, *Satisfiability coding lemma*, Proceedings of FOCS'97, 1997, pp. 566–574.

[Pap91] C. H. Papadimitriou, *On selecting a satisfying truth assignment*, Proceedings of FOCS'91, 1991, pp. 163–169.

[Pap94] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.

[Rob86] J. M. Robson, *Algorithms for maximum independent set*, Journal of Algorithms **7** (1986), 425–440.

[SAFS95] J. S. Schlipf, F. S. Annexstein, J. V. Franco, R. P. Swaminathan, *On finding solutions for extended Horn formulas*, Information Processing Letters **54 (3)** (1995), 133–137.

[Sch99] U. Schöning, *A probabilistic algorithm for k-SAT and constraint satisfaction problems*, Proceedings of FOCS'99, 1999, pp. 410–414.

[SLM92] B. Selman, H. Levesque, D. Mitchell, *A new method for solving hard satisfiability problems*, Proceedings of AAAI-92 (W. Swartout, ed.), MIT Press, 1992, pp. 440–446.

[Tod91] S. Toda, *PP is as hard as the polynomial-time hierarchy*, SIAM Journal on Computing **20(5)** (1991), 865–877.

[VV86] L. Valiant, V. Vazirani, *NP is as easy as detecting unique solutions*, Theoretical Computer Science **47** (1986), 85–93.

[Vse00] M. Vsemirnov, *Automorphisms of projective spaces and min-wise independent families of permutations*, Manuscript in preparation, 2000.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'