Journal of Universal Computer Science, vol. 12, no. 6 (2006), 725-745
submitted: 31/5/06, accepted: 23/6/06, appeared: 28/6/06 © J.UCS

Improving Deterministic and Randomized Exponential-Time Algorithms for the Satisfiability, the Colorability, and the Domatic Number Problem

Tobias Riege and **Jörg Rothe**
(Heinrich-Heine-Universität Düsseldorf, Germany
{riege, rothe}@cs.uni-duesseldorf.de)

Abstract: NP-complete problems cannot have efficient algorithms unless P = NP. Due to their importance in practice, however, it is useful to improve the known exponential-time algorithms for NP-complete problems. We survey some of the recent results on such improved exponential-time algorithms for the NP-complete problems satisfiability, graph colorability, and the domatic number problem. The deterministic time bounds are compared with the corresponding time bounds of randomized algorithms, which often run faster but only at the cost of having a certain error probability.

Key Words: Exact computation, exponential-time algorithms, deterministic algorithms, randomized algorithms
Category: F.1.2, F.1.3, F.2.2, F.2.3

1 Introduction

The design and analysis of algorithms is the core of computer science. Unfortunately, many problems that are important in practice are intractable—they currently do not have efficient (i.e., polynomial-time) algorithms. Among these problems, the NP-complete problems are particularly prominent: Unless P = NP, no NP-complete problem is in P. Here, P is the class of problems solvable in deterministic polynomial time, and NP contains all problems solvable in nondeterministic polynomial time.

The theory of NP-completeness was pioneered by Cook [Coo71], Karp [Kar72], and Levin [Lev73], see Garey and Johnson [GJ79] for an excellent guide to the theory of NP-completeness. To compare the complexity of two given problems, A and B, the polynomial-time many-one reducibility ($\leq_{\mathrm{m}}^{\mathrm{p}}$) is used: $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ if and only if there is a polynomial-time computable function f such that $x \in A$ if and only if $f(x) \in B$. A set B is NP-hard if and only if every NP set A $\leq_{\mathrm{m}}^{\mathrm{p}}$-reduces to B. If B is NP-hard and contained in NP, then B is said to be NP-complete. In this sense, NP-complete problems are the hardest problems in NP.

Plenty of problems have been shown NP-complete. In order to cope with the intractability of NP-complete problems in practice, various approaches have been proposed, among them parameterized complexity, approximation, and randomization. In this paper, we survey the recent progress on improving the known *exact* algorithms for such hard problems, both deterministic and randomized ones.

Since these are exponential-time algorithms, there is some constant $c > 1$ such that the algorithm runs in time $\tilde{\mathcal{O}}(c^n)$ in the worst case, where—as is common for exponential-time algorithms—the $\tilde{\mathcal{O}}$ notation is used to neglect polynomial factors: $\tilde{\mathcal{O}}(f(x)) = \mathcal{O}(\mathrm{poly}(x) \cdot f(x))$, where poly is a fixed polynomial. This definition is reasonable for exponential time. For example, if $n = 10,000$ then the running time of $n^2 \cdot 1.5^n$ lies between 1.5027^n and 1.5028^n.

Suppose we want to improve an exponential-time algorithm for some hard problem that runs in time $\tilde{\mathcal{O}}(c^n)$. Our goal then is to decrease the constant $c > 1$. For example, if the trivial algorithm running in time, say, $\tilde{\mathcal{O}}(3^n)$ can be improved to an algorithm running in time $\tilde{\mathcal{O}}\left(\left(\sqrt{3}\right)^n\right) \approx \tilde{\mathcal{O}}(1.732^n)$, then this improved algorithm can solve—in the same amount of time—problem instances twice as large as those solvable by the trivial algorithm, since $\left(\sqrt{3}\right)^{2n} = 3^n$. This difference can be quite important in practice.

We present both deterministic and randomized exponential-time algorithms for the NP-complete problems satisfiability, colorability, and the domatic number problem. While such algorithms for the first two problems have been intensely investigated since many years, there are rather few results for the domatic number problem. In each case, we merely give the rough idea of how the algorithms work. For other recent surveys on this topic, we refer to Schöning [Sch05], Woeginger [Woe03], and Fomin, Grandoni, and Kratsch [FGK05].

This survey is organized as follows. Section 2 presents exact algorithms for the satisfiability problem, and in particular for 3-`SAT`. Section 3 describes recent improvements on solving the graph colorability problem, again with a special focus on the case of 3-`Colorability`. The domatic number problem is dealt with in Section 4, and we again pay particular attention to the special problem 3-`DNP`. Finally, Section 5 summarizes the results presented and gives an outlook.

2 Satisfiability

A boolean formula $F = F(X, C)$ in conjunctive normal form (CNF) consists of a set $X = \{x_1, x_2, \ldots, x_n\}$ of boolean variables and of a set $C = \{c_1, c_2, \ldots, c_m\}$ of conjunctions of clauses over literals from X. A *literal* is either a variable $x_i \in X$ or its negation $\overline{x_i}$. For a given CNF formula F, the *satisfiability problem*, denoted by `SAT`, asks whether there exists a truth assignment $t : X \to \{0, 1\}$ such that every clause of C has at least one satisfied literal, i.e., t maps this literal to the value *true* (represented by 1). When the input is restricted to clauses with at most k literals each, this problem is called k-`SAT`.

Let t' be a partial truth assignment of the variables, where not all variables have been mapped to one of the values 0 or 1. The formula resulting from applying this partial assignment to formula F is denoted by $F_{|t'}$: All clauses of F containing at least one literal satisfied by t' can already be discarded and so do not occur in $F_{|t'}$, while literals of F set to *false* by t' are removed from their clauses in $F_{|t'}$. It is clear that if this procedure produces an empty clause, then the partial assignment t' cannot be extended to a satisfying truth assignment of F. On the other hand, if $F_{|t'}$ is the empty formula, then every complete extension of t' satisfies F.

Deciding whether a boolean formula is satisfiable was the first problem proven to be NP-complete by Cook [Coo71] and, independently, by Levin [Lev73]. The restricted version k-SAT remains NP-complete for $k \geq 3$, whereas 2-SAT is solvable in polynomial time. The naive algorithm for the satisfiability problem has a running time of $\tilde{\mathcal{O}}(2^n)$. It looks up every possible truth assignment t and verifies if t satisfies the boolean formula F. The worst-case time bounds for all exponential-time algorithms solving SAT in this survey are with respect to the number n of variables. There are only a few results where the number m of clauses is regarded as the size of the input. For example, Hirsch [Hir00] obtains a running time of $\tilde{\mathcal{O}}(1.2389^m)$, which was then improved by Yamamoto to a worst-case bound of $\tilde{\mathcal{O}}(1.2335^m)$ [Yam05]. Yamamoto's algorithm will be used in Section 4 for solving the domatic number problem. For the rest of this section, the restricted version 3-SAT will be analyzed, and all time bounds mentioned are with respect to the number n of boolean variables.

Starting with a simple idea for solving 3-SAT, one can improve the running time of the trivial algorithm to $\tilde{\mathcal{O}}(1.9130^n)$. Since each clause has at most three literals, one of the eight partial assignments of the three boolean variables cannot lead to a satisfied formula. For instance, if the clause $c = (\ell_1, \ell_2, \ell_3)$ is contained in the formula F, the case $\ell_1 = \ell_2 = \ell_3 = 0$ can be omitted. This technique is called "pruning the search tree," since dead ends of the computation tree are cut off once they have been detected. Processing the formula F clause by clause, this simple algorithm runs in time $\tilde{\mathcal{O}}((2^3 - 1)^{n/3}) = \tilde{\mathcal{O}}(1.9130^n)$.

Monien and Speckenmeyer were the first to give a better time bound [MS85]. Refining the above algorithm, they end up with a running time of $\tilde{\mathcal{O}}(1.6181^n)$ for 3-SAT. Most crucially, they notice that once a clause $c = (\ell_1, \ell_2, \ell_3)$ has been picked from the given formula F, it suffices to branch into the three cases $F_{|\ell_1=1}$, $F_{|\ell_1=0,\ell_2=1}$, and $F_{|\ell_1=\ell_2=0,\ell_3=1}$. The computation is split into cases having one, two, or three variable(s) fewer than before, which yields the recursion

$$T(n) \leq T(n-1) + T(n-2) + T(n-3).$$

Here, $T(n)$ is defined as the worst-case running time of the algorithm on an input with n variables left. Using standard methods, this bound is $T(n) = \alpha^n$, where $\alpha = 1.8393$ is the largest real root of the equation $\alpha^3 - \alpha^2 - \alpha - 1 = 0$. Monien and Speckenmeyer improve this idea even further by taking into consideration autark partial assignments, which are partial assignments t' that satisfy each clause in which any literal being assigned by t' occurs. Autarkness can be recognized in polynomial-time, and for every autark partial assignment of F, the formulas F and $F_{|t'}$ are equivalent. Hence, either the algorithm recurses into the (polynomially recognizable) case of autarkness, where at least one variable is eliminated, or a partial assignment is chosen which decreases the size of one remaining clause. This analysis leads to a worst-case time bound of $\tilde{\mathcal{O}}(1.6181^n)$.

By a deeper analysis of all the different subcases which can occur when inserting a partial assignment, the running time has been pushed down even further, see Schiermeyer [Sch92, Sch96b] and Kullmann [Kul97, Kul99]. In Table 1, these results are listed as well. Instead of analyzing dozens of subcases to reach better worst-case bounds, other ideas than backtracking will be discussed for the rest of this section.

Next, an algorithm proposed by Pudlák [Pud98] will be presented. Though it does not outperform the general backtracking algorithms, the way to search for a valid solution is quite nice. Rather than splitting the computation paths when assigning a truth value to a certain variable, Pudlák's idea is to divide the variable set X into half, and first solve the satisfiability problem for the clauses containing only variables from one of the two halves. This is a classical divide and conquer approach. Note that it is not quite obvious how to extend the divide process after the first level.

Let $F = F(X, C)$ be a given boolean formula in 3-CNF. Without loss of generality, assume that the number n of variables is even. Define $X_1 = \{x_1, \dots, x_{n/2}\}$ and $X_2 = \{x_{n/2+1}, \dots, x_n\}$ to be the halved variable sets. Let C_1 and C_2 be subsets of C whose clauses contain only variables from the sets X_1 and X_2, respectively. The rest of the clauses, in which variables from both X_1 and X_2 occur, are put into the set C_3. At first, the algorithm solves the two 3-SAT instances $F_1 = F_1(X_1, C_1)$ and $F_2 = F_2(X_2, C_2)$, which can be done with the naive algorithm in time $\tilde{\mathcal{O}}(2^{n/2}) = \tilde{\mathcal{O}}(1.4143^n)$. Let the set of satisfying truth assignments for F_1 and F_2 be called S_1 and S_2, respectively. Both sets are regarded as lists. If one of them is empty, the algorithm terminates, since then F is also unsatisfiable. Otherwise, for every $s \in S_1$, evaluate each clause $c \in C_3$ containing exactly two literals over X_1. Either one literal is true and c evaluates to true, or both literals are false, in which case the third literal over X_2 has to be set to true. The same is done for each partial assignment $s \in S_2$. In the next step, the algorithm tries to find a matching pair, $s_1 \in S_1$ and $s_2 \in S_2$, so that both partial assignments agree on every variable which has been set so far. Now, the trivial 2^n barrier can be beaten by constructing two separate methods, one of which performs well on partial assignments with many variables set to truth values, while the other one is relatively fast when truth values have been assigned to only a few variables. This way Pudlák achieves a running time of $\tilde{\mathcal{O}}(1.8487^n)$, which of course is not competitive to the 3-SAT algorithms examined so far.

Another drawback of Pudlák's method is that it is limited to the restricted case 3-SAT; by contrast, all other SAT-algorithms presented in this survey can be extended to solve the general k-SAT problem.

The last deterministic algorithm for the satisfiability problem discussed here is based on a local search method introduced by Schöning [Sch99]. Although this approach was originally used to design a randomized algorithm for the satisfiability problem, the method can be adapted to yield a deterministic algorithm by a few modifications, i.e., an algorithm that always finds the correct solution. This derandomization is described in [DGH$^+$02]. The algorithm was slightly improved in [BK04], which yields the currently best known time bound $\tilde{\mathcal{O}}(1.4726^n)$ for deterministic algorithms solving 3-SAT.

Suppose the boolean formula $F = F(X, C)$ is given and is satisfied by a truth assignment s. The local search method works as follows. To eventually reach this satisfying assignment, the local search method first guesses an initial truth assignment t at random. Define the Hamming distance d between s and t as the number of bits in which they differ. If t itself is a satisfying assignment, the algorithm outputs that F is satisfiable. Otherwise, F is not satisfied by t, therefore in at least one clause $c_i = (\ell_{i1}, \ell_{i2}, \ell_{i3})$ of C, all three literals are set to false by t. By recursing into the three subcases where the value of exactly one literal of c_i is flipped, the Hamming distance to the satisfying assignment s has been decreased by one for at least one branch of the recursion tree. If the Hamming distance between the initial assignment t and the satisfying assignment was d, this method would find s in time $\tilde{\mathcal{O}}(3^d)$. Starting from the two assignments 0^n and 1^n, the algorithm is within Hamming distance of at most $n/2$ of a possible satisfying truth assignment, so this simple deterministic procedure already has a worst-case running time of $\tilde{\mathcal{O}}(3^{n/2}) = \tilde{\mathcal{O}}(1.7321^n)$. Dantsin et. al [DGH$^+$02] further refined this idea by computing a suitable *covering code*, which is a set of initial assignments that, given an optimally chosen Hamming distance d, will reach every possible satisfying assignment $s \in \{0,1\}^n$ through the local search method. Deeper analysis of the number of starting assignments and the Hamming distance led to the currently best worst-case time bound of $\tilde{\mathcal{O}}(1.4726^n)$ for 3-SAT by Brueggemann and Kern [BK04].

Turning to randomized algorithms for 3-SAT, two basic probabilistic methods will be sketched. The first concept is based on a method by Paturi, Pudlák, and Zane [PPZ97]. Their simple algorithm first guesses a permutation π on the set $\{1, 2, \ldots, n\}$. Then, a random value from $\{0, 1\}$ is assigned to each bit of $x_{\pi(x)}$, and the formula is simplified by omitting clauses that contain satisfied literals and by dropping false literals, just as with evaluating partial assignments. At the end of the for-loop, the algorithm either ends up with an empty formula, in which case a satisfying truth assignment has been found, or it runs across an empty clause, which means that the formula is unsatisfiable, or the wrong random choices have been made. The error rate in the second case is at most

$1/2$ for each variable guess. Thus, the probability p to succeed in one cycle sums up to $p \geq (1/2)^n$ if formula F is satisfiable. Standard calculations show that repeating this process r times, where $r \geq \ln(1+\epsilon)/p$, leads to an error rate below ϵ. Not considering the simplifications during the probabilistic moves, this approach results in a running time of $\mathrm{poly}(n)/(1/2)^n \in \tilde{\mathcal{O}}(2^n)$. Note that this time bound is not better than the trivial 3-SAT bound. However, the expected running time can be considerably improved by checking whether there exists a unit clause $c = (x_i)$ or $c = (\overline{x_i})$ before each step i of the computation. In this case, the value of the variable x_i can be set to true or false, respectively, in a (correct) deterministic way, assuming that all the previous choices were correct. Paturi, Pudlák, and Zane [PPZ97] show that this boosts the success probability of one turn to $p \geq (1/2)^{(2n/3)}$, leading to a random algorithm of complexity $\mathrm{poly}(n) \cdot 2^{2n/3} \in \tilde{\mathcal{O}}(1.5875^n)$ for 3-SAT.

In a follow-up paper, Paturi, Pudlák, Saks, and Zane [PPSZ05] improve this bound even further to $\tilde{\mathcal{O}}(1.3632^n)$. The clauses of the formula F can be preprocessed before each random step, making the occurance of unit clauses, and therefore deterministic choices, much more likely. The decreased rate of possible wrong guesses then straightforwardly leads to a smaller number r of repetitions needed in the algorithm.

The second probabilistic approach is called randomized walk, and it is closely related to the deterministic local search method presented above. The idea is due to Schöning [Sch99]. Just as with local search, the random walk algorithm starts with an initial random assignment $t \in \{0,1\}^n$ for the formula $F = F(X, C)$. If t does not satisfy F, an unsatisfied clause $c = (\ell_1, \ell_2, \ell_3)$ is chosen, but instead of splitting into the three subcases by setting exactly one literal of c to true, the variable to be flipped is chosen at random. If after $3n$ bit flips no satisfying truth assignment has been found, the process is restarted with another random initial assignment t'. In order to determine the number r of repetitions needed, the error rate for each loop has to be bounded. Since the probabilistic algorithm can make errors only if the given formula F is satisfiable, suppose that F is satisfiable via some assignment s. The probability that the Hamming distance between s and the initial guess $t \in \{0,1\}^n$ is d equals $\binom{n}{d} \cdot 2^{-n}$. As each clause consists of three literals, the probability of decreasing the Hamming distance with a bit flip is at least $1/3$, whereas an increase can be as likely as $2/3$. Still, if the initial guess is close to the satisfying assignment s, there is a high probability that the algorithm will find s. The bit flipping continues for $3n$ steps, so that some wrong decisions when picking a literal can be compensated for later. The success rate for each random walk can be shown to be $(3/4)^n$, so after $r = (4/3)^n$ iterations the overall error rate is negligibly small. The result is a probabilistic algorithm running in time $\tilde{\mathcal{O}}(1.3334^n)$. Actually, Papadimitriou first used this random walk method to construct a probabilistic algorithm solving 2-SAT in

Source	Type	Bound
trivial algorithm	deterministic & randomized	2^n
simple backtracking	deterministic	1.9130^n
Pudlák [Pud98]	deterministic	1.8487^n
Monien and Speckenmeyer [MS85]	deterministic	1.6181^n
Schiermeyer [Sch96b]	deterministic	1.4970^n
Kullmann [Kul99]	deterministic	1.4963^n
Dantsin et al. [DGH$^+$02]	deterministic	1.4802^n
Brueggemann and Kern [BK04]	deterministic	1.4726^n
Paturi, Pudlák, and Zane [PPZ97]	randomized	1.5875^n
Paturi et al. [PPSZ05]	randomized	1.3632^n
Schöning [Sch99]	randomized	1.3334^n
Hofmeister et al. [HSSW02]	randomized	1.3302^n
Baumer and Schuler [BS03]	randomized	1.3290^n
Iwama and Tamaki [IT04]	randomized	1.3238^n
Rolf [Rol05]	randomized	1.3222^n

Table 1: Worst-case time bounds for 3-SAT

time $\mathcal{O}(n^2)$ [Pap91]. Since the formula in 2-CNF has at most two literals in each clause, the error rate of the random choice can be bounded by 1/2.

Turning back to the NP-complete problem 3-SAT, the above bound was improved to $\tilde{\mathcal{O}}(1.3302^n)$ by a clever idea of Hofmeister et al. [HSSW02]. Before taking the initial guess, they collect a set of m variable-disjoint clauses. Depending on the number m, the algorithm splits into two cases. On the one hand, if m is very small, each of the 7^m potential partial truth assignments t are tested,[1] leaving a formula $F_{|t}$ in 2-CNF, which can be solved in polynomial time. On the other hand, if m exceeds a certain ratio (namely, $m \geq 0.1469n$), all variables are set at random as before, with the difference that the probability distribution is now depending on the input formula. Each set of three variables contained in one of the m clauses c collected in the first step is now set to specific values as follows: either exactly one, or two, or three literals are set to true, with the best probabilities for each of these three cases yet to be determined. The partial assignment that leaves c unsatisfied is not considered, which increases the probability to find a satisfying assignment. Note that the number of biased random choices increases proportional to m, since only variables not contained in one of the m clauses are set to true or false with probabilty exactly 1/2. Altogether,

[1] For each clause $c = (\ell_1, \ell_2, \ell_3)$, one of the eight partial assignments mapping the literals ℓ_1, ℓ_2, and ℓ_3 to truth values never leads to a satisfying truth assignment, namely $t(\ell_1) = t(\ell_2) = t(\ell_3) = 0$.

the algorithm reaches a worst-case time bound of $\tilde{\mathcal{O}}(1.3302^n)$.

Iwama and Tamaki [IT04] proposed a clever combination of the algorithms of Schöning [Sch99] and Paturi et al. [PPSZ05] to obtain a worst-case time bound of $\tilde{\mathcal{O}}(1.3238^n)$. Their algorithm was further improved by Rolf [Rol05], whose algorithm with a time bound of $\tilde{\mathcal{O}}(1.3222^n)$ is the current champion of the probabilistic methods to solve 3-`SAT`. Table 1 compares the worst-case running times of the presented deterministic and probabilistic algorithms solving 3-`SAT`.

3 Colorability

A graph is a pair $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of vertices and $E = \{\{v_i, v_j\} \mid 1 \leq i < j \leq n\}$ is a set of edges, each edge connecting two vertices. Two connected vertices are called adjacent. A mapping f from V to the set $\{1, 2, \ldots, k\}$ is called a *k-coloring of G*. A coloring is said to be *legal* if and only if no two adjacent vertices are mapped to the same color. The minimum number k such that G has a legal k-coloring is called the *chromatic number of* G and is denoted by $\chi(G)$. One may look at a coloring $f : V \to \{1, \ldots, k\}$ of graph G as a partition of the vertex set V into k disjoint sets $V_1, \ldots, V_k$, where each set represents a color class $V_i = \{v \in V \mid f(v) = i\}$. In the case of a legal coloring, each set of the partition must be an independent set, i.e., must contain only nonadjacent vertices. Given a graph G, the general colorability problem asks to determine the value of $\chi(G)$. The set k-`Colorability` contains all graphs G with $\chi(G) \leq k$. A graph G can be legally colored with two colors if and only if it is bipartite, thus 2-`Colorability` is in P. It is known that k-`Colorability` is NP-complete for all $k \geq 3$, see [Kar72].

The first nontrivial exact exponential-time algorithm for the colorability problem is based on dynamic programming and is due to Lawler [Law76]. The idea comes from a simple observation. A *maximal* independent set is an independent set that is no proper subset of another independent set. Among all the legal colorings f of G that are minimal (that is, f maps to exactly $\chi(G)$ colors), at least one has to contain such a maximal independent set. To see this, take one of the minimal colorings and one specific independent set U of the partition, which is mapped to color i. If U itself is not maximal, there exists a set U' with $U \subset U'$ and U' is a maximal independent set. Change f to f' by assigning to each vertex in U' the color i. Then f' is a legal coloring with $\chi(G)$ colors containing a maximal independent set.

With this in mind, an algorithm calculating $\chi(G)$ can easily be constructed. For a graph $G = (V, E)$ and any subset $V' \subseteq V$, denote by $G[V'] = (V', E')$ with $E' = \{\{u, v\} \in E \mid u, v \in V'\}$ the subgraph induced by V'. If the chromatic number of *all* induced subgraphs $G[V'']$ for every proper subset $V'' \subset V'$ has already been computed, $\chi(G[V'])$ is determined by the formula

$$\chi(G[V']) = 1 + \min\{\chi(G[V' - V''])\},$$

where the minimum is taken over all $V'' \subset V'$ such that V'' is a maximal independent set. The dynamic programming approach computes the chromatic number of all subsets of V with increasing cardinality. This way, during each iteration with a subset V', all chromatic numbers of $G[V'']$ with $V'' \subset V'$ are already known. The overall running time is $\sum_{k=1}^{n} \binom{n}{k} 2^k \in \tilde{\mathcal{O}}(3^n)$. Lawler improves this bound by using two results from graph theory. On the one hand, it is known that there are at most $3^{n/3}$ maximal independent sets for a graph with n vertices, which was proven by Moon and Moser [MM65]. On the other hand, each of these sets can be generated in time $\mathcal{O}(n^2)$, as was shown by Paull and Unger [PU59]. This leads to a worst-case time of

$$\sum_{k=1}^{n} \binom{n}{k} k^2 3^{k/3} \leq n^2 (1 + 3^{1/3})^n,$$

which is in $\tilde{\mathcal{O}}(2.4423^n)$. This result on the general colorability problem is further refined by Eppstein [Epp03] to $\tilde{\mathcal{O}}(2.4150^n)$.

Now turning to the special case of `3-Colorability`, the naive algorithm runs in time $\tilde{\mathcal{O}}(3^n)$. It cycles through all possible mappings from the n vertices to the three colors and verifies their legality. This naive approach is already beaten by Lawler's algorithm for the general colorability problem. The barrier $\tilde{\mathcal{O}}(2^n)$ is reached by picking a starting vertex $v_0 \in V$, assigning it to the first color set, and processing through the rest of the graph by a simple searching procedure like breadth-first search. Since every vertex is adjacent to a vertex that has already been mapped to one color, only two choices are left in each step. Hence, this splitting algorithm runs in time $\tilde{\mathcal{O}}(2^n)$.

Another simple idea improves this bound even further. If V is partitioned into three sets, the smallest set must have cardinality less than or equal to $n/3$. There are only $\sum_{k=1}^{n/3} \binom{n}{k/3}$ such sets. Listing all of them and checking that they are independent sets can be done in time $\tilde{\mathcal{O}}(1.8899^n)$. If the first set of the partition is given, say V_1, the remaining vertices in V can be legally two-colored in polynomial time. This task is the same as checking whether the induced subgraph $G[V - V_1]$ is bipartite, i.e., can be partitioned into two independent sets. With the ideas of Lawler, one can achieve a worst-case bound of $\tilde{\mathcal{O}}(1.4423^n)$ for determining if a given graph $G = (V, E)$ is legally three-colorable. Instead of looking at every subset, simply list all maximal independent sets $V' \subseteq V$. Their vertices are mapped to the first color class. The remaining problem is again solvable in polynomial time. As seen above, the time needed to generate all maximal independent sets is $n^2 \cdot 3^{n/3} \in \tilde{\mathcal{O}}(1.4423^n)$. Schiermeyer further improved this bound [Sch96a]. His method first chooses some vertex of maximum degree and continues with properly handling the many subcases that can occur, which requires a rather deep, technical analysis. The worst-case time bound given in his paper is $\tilde{\mathcal{O}}(1.3977^n)$.

Beigel and Eppstein [BE95] continued the progress of improving the running times of algorithms solving 3-`Colorability` with an utterly new approach. Instead of listing all possible independent sets for the partition and verifying each choice, they look at the coloring problem as a special case of a *constraint satisfaction problem*, `CSP` for short. This problem depends on two parameters, a and b, and is defined as follows. An (a, b)-`CSP` instance consists of a set of n variables, each of which can take on a distinct values, and additionally of m constraints, each involving at most b variables. Such an instance is solvable if and only if there exists a mapping from the n variables to the a values such that all constraints are satisfied. For example, the problem $(2, 3)$-`CSP` is nothing other than 3-`SAT`, since every variable may be set to one of the two values 1 (*true*) or 0 (*false*), and every clause represents the constraint that not all three literals in the clause are set to *false*. Similarly, colorability problems can be viewed as (a, b)-`CSP` for suitable a and b. For example, 3-`Colorability` can be regarded as a special case of $(3, 2)$-`CSP`. The vertices of the given graph represent the variables, which can be mapped to three distinct colors, and every edge induces the constraint that its two vertices cannot be mapped to the same color.

Beigel and Eppstein state and prove some useful basic properties of (a, b)-`CSP`. For instance, there is an easy transformation from every (a, b)-`CSP` instance to an equivalent (b, a)-`CSP` instance, which implies some form of duality. Even more importantly for the colorability problem, they show how to remove variables from $(a, 2)$-`CSP` instances that are restricted to only two of the a possible values. This result is used in the analysis of the following very simple randomized algorithm for 3-`Colorability`: For each vertex $v \in V$, randomly choose one of the three colors to which v *cannot* be mapped. With probability at least $2/3$, a random choice is correct, so the overall success probability is $(2/3)^n$. The resulting `CSP` instance is then equivalent to a 2-`SAT` formula, which can be solved in polynomial time. The reciprocal value of the success probability leads to a time bound of $\tilde{\mathcal{O}}(1.5^n)$ for this randomized algorithm for 3-`Colorability`. The same bound was achieved by Schöning [Sch99], who used the local search method described in Section 2 to solve the general constraint satisfaction problem.

To improve the above time bound of this simple probabilistic method, the following idea is used in [BE95]. Suppose there is a constraint involving two variables (in this case vertices), v_1 and v_2, such that both cannot share some value $a \in \{1, 2, 3\}$. Now restrict the color choices for the vertices v_1 and v_2 to one of the four cases where one color is eliminated for each vertex, such that *exactly one* vertex is allowed to be colored with b or $c \in \{1, 2, 3\}$, where $b \neq a \neq c$. It can be verified that this random choice is correct in half of the cases that can occur. For example, if one of the vertices, say v_1, needs to be mapped to color a, in two of the four cases color a was not eliminated for the correct vertex v_1. And as another example, if a must not be used in the coloring of v_1 and v_2, in half of the four cases the right color was kept as a choice. This way two vertices can be removed in one step by the simplifying process mentioned above, even though the error rate increases from $1/3$ to $1/2$. Still, the overall success rate is $(1/2)^{n/2}$, and so the randomized algorithm needs to be repeated only $\tilde{\mathcal{O}}(2^{n/2})$ times to reach a negligibly small error rate. This leads to a running time of $\tilde{\mathcal{O}}(1.4143^n)$.

This local elimination of colors for some set of vertices can be used to yield a deterministic algorithm for 3-`Colorability`, which has an even better running time than the probabilistic methods described so far. Suppose the $(3,2)$-`CSP` instance contains a variable v which, associated with a value $a \in \{1,2,3\}$, is involved in constraints with many other variables. The computation handles the following two cases sequentially: first, v is mapped to a; second, v is restricted to the other two values remaining. This is a deterministic algorithm, since both cases will be handled eventually. In the first case, many variables are restricted to two values because they were contained in constraints including v and a. Remember that in $(3,2)$-`CSP`, constraints consist of at most two variables. When restricted to two values, the variables can be eliminated by the standard deterministic procedure described in [BE95]. In the other case, only v can be removed from the instance, but this is balanced by the first branch, which reduces the `CSP` considerably. For the $(3,2)$-`CSP` instance corresponding to a 3-`Colorability` graph, the number of constraints is bounded by the degrees of the vertices. Vertices of degree two or less do not have to be considered in the case of 3-`Colorability`, and the recursion $T(n) \leq T(n-4) + T(n-1)$ is obtained, which gives a worst-case time bound of $\tilde{\mathcal{O}}(1.3803^n)$ for the algorithm. This bound was also reached for the general $(3,2)$-`CSP`, though the case analysis which leads to the same recursion as above is much more sophisticated.

An improvement to $\tilde{\mathcal{O}}(1.3446^n)$ is achieved by yet another idea of the same authors [BE95]: Pick a small set of vertices $V' \subset V$ such that most of the graph is covered by the open neighborhood

$$N(V') = \{v \in V - V' \mid \exists w \in V' \text{ with } \{v,w\} \in E\}$$

of V'. Now, all the $3^{|V'|}$ possible colorings for the vertices in V' are tested. For each mapping, every vertex $v \in N(V')$ is restricted to at most two colors, so they can be eliminated again by the standard procedure. The remaining $(3,2)$-`CSP` instance containing only vertices in $V - (V' \cup N(V'))$ is then solved with the algorithm described in the preceding paragraph.

Introducing an idea that has later been dubbed "measure and conquer," Eppstein [Epp01] improved the running time even further to $\tilde{\mathcal{O}}(1.3289^n)$. This new technique will be discussed in Section 4 for the domatic number problem.

Source	Type	Bound
trivial algorithm	deterministic & randomized	3^n
simple BFS	deterministic	2^n
Lawler [Law76]	deterministic	1.4422^n
Schiermeyer [Sch96a]	deterministic	1.3977^n
Beigel and Eppstein [BE95]	deterministic	1.3446^n
Eppstein [Epp01]	deterministic	1.3289^n
Schöning [Sch99]	randomized	1.5^n
Beigel and Eppstein [BE05]	randomized	1.4143^n

Table 2: Worst-case time bounds for 3-`Colorability`

Eppstein's bound is currently the best known result for recognizing if a given graph is colorable with three colors.

Table 2 lists all results for the 3-`Colorability` problem presented here. Interestingly, the randomized methods developed so far have worse running times than the best deterministic ones. This might be explained by the fact that the randomized algorithms for 3-`Colorability` are all very simple up to date, whereas the deterministic methods to color a graph with three colors have been studied and refined for quite a long time. Another reason for the rather sparse number of randomized algorithms for 3-`Colorability` might be that all random approaches solve the general constraint satisfaction problem, but colorability is actually only a special case of a `CSP`. The field of randomized coloring algorithms remains a promising research area for future work.

4 Domatic Number Problem

Let $G = (V, E)$ be a given graph. A subset $D \subseteq V$ of the vertex set is called a *dominating set of* G if and only if every vertex $u \in V - D$ is adjacent to at least one vertex $v \in D$. The *domatic number of* G, which is defined as the maximum number of disjoint dominating sets, is denoted by $\delta(G)$. The (search version of the) *domatic number problem* asks to determine the domatic number of G. In other words, G is to be partitioned into as many sets as possible that each are dominating. Defining the decision version of this problem, let k-`DNP` denote the set of graphs with $\delta(G) \geq k$. Note that for each $k \geq 3$, k-`DNP` is NP-complete, see Garey and Johnson [GJ79]. We here restrict our attention to 3-`DNP`.

Although the domatic number problem is important for various practical applications (for example, for the task of distributing resources in a network), there exist only very few exact exponential-time algorithms for 3-`DNP` in the literature.

It is not quite obvious why other interesting NP-complete graph problems—such as 3-Colorability—have been studied much more intensely up to date. One reason for this might be that the domatic number problem seemingly is more difficult to handle than the colorability problem, in the sense that "dominance" is a *global* property, whereas "independence" (and thus "colorability") has a *local* character. To wit, checking if a given set S is independent can be achieved by looking at its neighborhood, $N(S)$. In contrast, verifying that S is a dominating set is possible only when taking a look at the whole graph G. This difficulty can also be seen when transforming the domatic number problem into a constraint satisfaction problem: The question of whether a graph is colorable with k colors can be translated into a $(k, 2)$-CSP instance, as seen in Section 3, but k-DNP can be generalized only to a $(k, d+1)$-CSP instance, where $d = \Delta(G)$ is the maximum degree of the graph G. For each vertex $v \in V$, there are k constraints representing the condition that at least one of the vertices in $N[v]$, the closed neighborhood of v, needs to be part of one of the k dominating sets.

As noted in [RR05], Lawler's algorithm [Law76] for the colorability problem can be adapted to find the domatic number of a graph G in time $\tilde{\mathcal{O}}(3^n)$. This observation straightforwardly leads to an $\tilde{\mathcal{O}}(3^n)$ algorithm for k-DNP (and, in particular, for 3-DNP). While the bound for 3-Colorability could be improved by using results from graph theory, no similarly useful theorems helping to improve the worst-case bound for determining the domatic number are known until today.

The first algorithm breaking the trivial 3^n barrier is due to Riege and Rothe [RR05]. Their backtracking algorithm successively assigns the vertices in V to the three potential dominating sets in some kind of greedy manner, seeking to find vertices that seem to be most helpful in finding the three dominating sets. Those are usually the vertices of large degree, since they can dominate a larger part of the graph. It can be shown that this recursive procedure has a worst-case running time of $\tilde{\mathcal{O}}(2.9416^n)$, which is only slightly better than the trivial time bound.

A new approach is used by Fomin et al. [FGPS05]. Their "measure and conquer" technique is based on Eppstein's observation that improvements on the worst-case running time of an algorithm can be made by a more careful analysis of the backtracking procedures [Epp04]. Standard worst-case analysis often uses a too rough measurement when calculating the recurrences modeling the recursion. The common size for a problem instance is the number of variables (for satisfiability) or the number of vertices (for domatic number or colorability). This sometimes does not reflect the actual progress of the algorithm when processing different choices in the recursive branching. For example, mapping variables that very often occur in many clauses of a SAT instance to a specific value might have a greater impact than assigning a value to a less frequent variable. A finer analysis can be made by putting different weights to the variables, depending on the impact they have when being processed by the algorithm. This technique of further bounding the number of leaves in the computation tree can drastically improve the worst-case running time of one and the same algorithm. The optimum weights yielding the minimum upper bound of the backtracking algorithm can be found efficiently by Eppstein's quasiconvex programming method [Epp04].

Fomin et al. [FGPS05] use this approach to obtain an algorithm that lists all minimal dominating sets of a graph G with n vertices in time $\tilde{\mathcal{O}}(1.7697^n)$. They then use this result to design an algorithm for computing the domatic number of G in the following way. As was the case with maximal independent sets in minimal colorings of a graph, note that at least one of the partitions of G into the maximum number of dominating sets contains a minimal dominating set. For any subset $V' \subseteq V$, let $\delta'(G[V'])$ be the maximum number of disjoint minimal dominating sets of G containing only vertices in V'. With basically the same idea that Lawler used to compute the chromatic number of G, one can dynamically construct the domatic numbers of induced subgraphs, using

$$\delta'(G[V']) = \max\{\delta'(G[V' - D]) + 1\},$$

where the maximum is taken over all minimal dominating sets $D \subseteq V'$ of G. Via Eppstein's numerical procedure [Epp04], one can optimize the parameters of the occurring recurrences and reach a worst-case time of $\tilde{\mathcal{O}}(2.8805^n)$ to obtain the domatic number of the given graph G. Once $\delta(G)$ is known, membership in 3-`DNP` is easily determined.

The worst-case bound of several simple algorithms for NP-hard problems could be improved significantly at the cost of having to deal with a more complex recursive analysis. Fomin, Grandoni, and Kratsch recently published a survey of the "measure and conquer" method [FGK05], and they also discuss lower bounds for exact exponential-time algorithms. Knowing the lower bound for a specific algorithm gives an indication of how far away from the best bound possible a proven worst-case upper bound might be.

The currently best worst-case time of a 3-`DNP` algorithm running in polynomial space is due to Riege, Rothe, Spakowski, and Yamamoto [RRSY, RRSY06]. Their procedure first lists all minimal dominating sets of a given graph G, which is possible in time $\tilde{\mathcal{O}}(1.7697^n)$ as shown by Fomin et al. [FGPS05]. For each such set D, it remains to find two other dominating sets. This problem can be transformed to a special case of the satisfiability problem. For each vertex $v \in V$, create a clause containing every vertex in the closed neighborhood $N[v]$ which is not already contained in D. Then, try to solve this `SAT` instance in the not-all-equal sense, meaning that at least one, but not all of the literals in each clause must be set to true. By doubling the number of clauses, this not-all-equal

Source	Type	Bound
trivial algorithm	det. & rand.	3^n
Riege and Rothe [RR05]	deterministic	2.9416^n
Fomin et al. [FGPS05]	deterministic	2.8805^n
Riege, Rothe, Spakowski, Yamamoto [RRSY, RRSY06]	deterministic	2.6949^n

Table 3: Deterministic worst-case time bounds for 3-DNP

SAT instance can be translated to a regular SAT formula, which is then solved in time $\tilde{\mathcal{O}}(1.234^m)$ with Yamamoto's algorithm [Yam05]. Here, m is the number of clauses. Since in this case m equals $2n$, where n is the number of vertices in graph G, the overall time-bound is $\tilde{\mathcal{O}}(1.7697^n \cdot 1.234^{2n}) = \tilde{\mathcal{O}}(2.6949^n)$.

Table 3 lists all currently known bounds of deterministic 3-DNP algorithms running in polynomial space. We mention that Björklund and Husfeldt [BH06] obtained better worst-case time bounds for algorithms not running in polynomial space. In particular, they presented an algorithm that finds the domatic number of a given graph in time and space $\tilde{\mathcal{O}}(2^n)$.

Randomized algorithms for 3-DNP have been studied only for input graphs of bounded maximum degree. Two results have been obtained and are summarized in Table 4 for graphs with maximum degree $\Delta(G)$, $3 \leq \Delta(G) \leq 7$. The performance of these two randomized algorithms is compared to the following simple deterministic method [RR05]. First, process the graph by breadth-first search. For each vertex v of the vertex set V, branch into all cases that assign all vertices in the closed neighborhood $N[v]$ to the three dominating sets, looking only at cases which might lead to a valid partition. For example, the case of assigning all vertices of $N[v]$ into one set will not be considered. Here, the number of possible choices increases with the degree of the vertex and hence the running time depends on the maximum degree of the given graph G.

The first randomized algorithm essentially uses the same method [RR05]. Instead of trying all valid partitions of the closed neighborhood $N[v]$, pick three vertices in $N[v]$ at random and assign them to the three different dominating sets. The second random algorithm [RRSY, RRSY06] uses the probabilistic method of Schöning [Sch99] to solve constraint satisfaction problems. Note that an instance of 3-DNP restricted to graphs G with maximum degree $d = \Delta(G)$ can be transformed into a $(3, d+1)$-CSP instance. The second randomized procedure performs better than the first one whenever $\Delta(G) \geq 5$.

Source	Type	$\Delta(G) = 3$	4	5	6	7
[RR05]	deterministic	2.2894^n	2.6591^n	2.8252^n	2.9058^n	2.9473^n
[RR05]	randomized	2^n	2.3570^n	2.5820^n	2.7262^n	2.8197^n
[RRSY, RRSY06]	randomized	2.2501^n	2.4001^n	2.5001^n	2.5715^n	2.6251^n

Table 4: Results for 3-`DNP` on graphs with bounded maximum degree

5 Summary

In this paper, we have surveyed various exact exponential-time algorithms for the NP-complete problems 3-`SAT`, 3-`Colorability`, and 3-`DNP`, trying to document the progress that has been made at improving the worst-case time bounds since the first such algorithms appeared in the early seventies. Additionally, the results are compared to the upper bounds of randomized algorithms for these problems. In the case of the satisfiability and the domatic number problem, the power of randomness can significantly reduce the running time of the algorithms, at the cost of making errors with a certain probability. Using standard probability amplification, this error can be made negligibly small.

We mention that exact exponential-time algorithms have also been proposed for NP-hard problems other than those presented here. For example, constraint satisfaction problems were considered by Kumar [Kum92] and Schöning [Sch99]. One of the most famous examples of an NP-complete set, the traveling salesperson problem, still withstands all attempts to break the $\tilde{\mathcal{O}}(2^n)$ barrier, which nonetheless is much better than the trivial method of skimming through all the $n!/2$ possible tours. Exact algorithms for the NP-complete independent set problem are already very close to the barrier $c = 1$ in $\tilde{\mathcal{O}}(c^n)$, which would imply P = NP and so is very unlikely ever to be reached. The currently best independent set algorithm is due to Robson [Rob01] and achieves $c = 1.1889$ in the worst-case (see also [Rob86] for previous results on finding maximum independent sets in a graph).

There are problems which lie between the complexity classes P and NP (assuming P $\neq$ NP) in the sense that they clearly belong to NP, yet have resisted any attempt to show NP-hardness as well as any attempt to find a polynomial-time algorithm. Until recently, the question of whether a given integer is prime was one of those problems. By very clever arguments from number theory, Agrawal, Kayal, and Saxena found the first deterministic algorithm solving this ancient problem in polynomial time [AKS04]. Another problem which seems to be neither in P nor NP-complete is the graph isomorphism problem, `GI` for short. This problem contains all pairs of isomorphic graphs. Two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, are isomorphic if and only if there exists an edge-preserving permutation π mapping from V_1 to V_2, which means that $\{u, v\} \in E_1$ if and only if $\{\pi(u), \pi(v)\} \in E_2$. This problem has been intensely studied, see Köbler, Schöning, and Torán [KST93]. Although GI has not been proven to be NP-hard so far, no efficient algorithm finding an isomorphism between two given graphs is known. The naive procedure that cycles through all possible permutations runs in time $\tilde{\mathcal{O}}(n!)$. To our knowledge, the method of Babai is the only known exact algorithm besides the trivial one solving the graph isomorphism problem [Bab81]. This algorithm runs in time $\tilde{\mathcal{O}}(e^{n^{1-c}})$ with $c = 1/3 + o(1)$.

In practice, some NP-complete problems turn out to be efficiently solvable on the average. Average-case complexity deals with this phenomenon, and Levin was one of the first trying to capture this notion in a mathematical precise way [Lev86]. Given a certain probability distribution on the inputs, one can define what it means that an algorithm performs well on the average. One example of an NP-complete problem that is easy to solve on the average is the graph coloring problem [Wil84]. However, a lot of problems remain hard to solve even in the average-case model. For surveys of average-case complexity, the reader is referred to Wang [Wan97a, Wan97b], Goldreich [Gol97], and Ben-David et al. [BCGL92].

Acknowledgments

This work was supported in part by the German Science Foundation under grants RO 1202/9-1 and RO 1202/9-3 and by the Alexander von Humboldt Foundation in the CONNECT program. The second author thanks Osamu Watanabe for his kind hospitality at the Tokyo Institute of Technology, where some results reported on here were obtained.

References

[AKS04] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[Bab81] L. Babai. Moderately exponential bound for graph isomorphism. In *Proceedings of the International Conference on Fundamentals of Computation Theory*, pages 34–50. Springer-Verlag *Lecture Notes in Computer Science #117*, August 1981.

[BCGL92] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44(2):193–219, 1992.

[BE95] R. Beigel and D. Eppstein. 3-coloring in time $\mathcal{O}(1.3446^n)$: A no-MIS algorithm. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 444–452, 1995.

[BE05] R. Beigel and D. Eppstein. 3-coloring in time $\mathcal{O}(1.3289^n)$. *Journal of Algorithms*, 54(2):168–204, 2005.

[BH06] A. Björklund and T. Husfeldt. Inclusion-exclusion algorithms for counting set partitions. In *Proceedings of the 47th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, October 2006. To appear.

[BK04] T. Brueggemann and W. Kern. An improved deterministic local search algorithm for 3-SAT. *Theoretical Computer Science*, 329(1-3):303–313, 2004.

[BS03] S. Baumer and R. Schuler. Improving a probabilistic 3-SAT algorithm by dynamic search and independent clause pairs. In *6th International Conference on Theory and Applications of Satisfiability Testing*, pages 150–161. Springer-Verlag *Lecture Notes in Computer Science #2919*, 2003.

[Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, pages 151–158. ACM Press, 1971.

[DGH+02] E. Dantsin, A. Goerdt, E. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, October 2002.

[Epp01] D. Eppstein. Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 329–337. Society for Industrial and Applied Mathematics, 2001.

[Epp03] D. Eppstein. Small maximal independent sets and faster exact graph coloring. *Journal of Graph Algorithms and Applications*, 7(2):131–140, 2003.

[Epp04] D. Eppstein. Quasiconvex analysis of backtracking algorithms. In J. Munro, editor, *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms*, pages 788–797. Society for Industrial and Applied Mathematics, 2004.

[FGK05] F. Fomin, F. Grandoni, and D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS*, 87:47–77, 2005.

[FGPS05] F. Fomin, F. Grandoni, A. Pyatkin, and A. Stepanov. Bounding the number of minimal dominating sets: A measure and conquer approach. In *Proceedings of the 16th International Symposium on Algorithms and Computation*, pages 573–582, 2005.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[Gol97] O. Goldreich. Note on Levin's theory of average-case complexity. Technical Report TR97-058, Electronic Colloquium on Computational Complexity, November 1997.

[Hir00] E. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24:397–420, 2000.

[HSSW02] T. Hofmeister, U. Schöning, R. Schuler, and O. Watanabe. A probabilistic 3-SAT algorithm further improved. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, pages 192–202. Springer-Verlag *Lecture Notes in Computer Science #2285*, 2002.

[IT04] K. Iwama and S. Tamaki. Improved upper bounds for 3-SAT. In J. Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 328–329. Society for Industrial and Applied Mathematics, January 2004.

[Kar72] R. Karp. Reducibilities among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, 1972.

[KST93] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity.* Birkhäuser, 1993.

[Kul97] O. Kullmann. Worst-case analysis, 3-SAT decision and lower bounds: Approaches for improved SAT algorithms. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 261–313. American Mathematical Society, 1997.

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

[Kum92] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[Law76] E. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66–67, 1976.

[Lev73] L. Levin. Universal sorting problems. *Problemy Peredaci Informacii*, 9:115–116, 1973. In Russian. English translation in *Problems of Information Transmission*, 9:265–266, 1973.

[Lev86] L. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

[MM65] J. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965.

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

[Pap91] C. Papadimitriou. On selecting a satisfying truth assignment. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, pages 163–169, 1991.

[PPSZ05] R. Paturi, P. Pudlák, M. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *Journal of the ACM*, 52(3):337–364, 2005.

[PPZ97] R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 566–574, 1997.

[PU59] M. Paull and S. Unger. Minimizing the number of states in incompletely specified state machines. *IRE Transactions on Electronic Computers*, EC-8:356–367, 1959.

[Pud98] P. Pudlák. Satisfiability – algorithms and logic. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science*, pages 129–141. Springer-Verlag *Lecture Notes in Computer Science #1450*, August 1998.

[Rob86] J. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, December 1986.

[Rob01] J. Robson. Finding a maximum independent set in time $\mathcal{O}(2^{n/4})$. Technical Report TR 1251-01, LaBRI, Université Bordeaux I, 2001. Available on-line at http://dept-info.labri.fr/~robson/mis/techrep.html.

[Rol05] D. Rolf. Improved bound for the PPSZ/Schöning-algorithm for 3-SAT. Technical Report TR05-159, Electronic Colloquium on Computational Complexity, 2005.

[RR05] T. Riege and J. Rothe. An exact 2.9416^n algorithm for the three domatic number problem. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, pages 733–744. Springer-Verlag *Lecture Notes in Computer Science #3618*, August 2005.

[RRSY] T. Riege, J. Rothe, H. Spakowski, and M. Yamamoto. An improved exact algorithm for the domatic number problem. *Information Processing Letters.* Accepted subject to revision.

[RRSY06] T. Riege, J. Rothe, H. Spakowski, and M. Yamamoto. An improved exact algorithm for the domatic number problem. In *Proceedings of the Second IEEE International Conference on Information & Communication Technologies: From Theory to Applications*, pages 1021–1022. IEEE Computer Society Press, April 2006. A six-page extended abstract is available from CD-ROM.

[Sch92] I. Schiermeyer. Solving 3-satisfiability in less than 1.579^n steps. In *Proceedings of the 6th Workshop on Computer Science Logic*, pages 379–394. Springer-Verlag *Lecture Notes in Computer Science #702*, 1992.

[Sch96a] I. Schiermeyer. Fast exact colouring algorithms. In *Tatra Mountains Mathematical Publications*, volume 9, pages 15–30, 1996.

[Sch96b] I. Schiermeyer. Pure literal look ahead: An $\mathcal{O}(1.497^n)$ 3-satisfiability algorithm. In *Proceedings of the Workshop on the Satisfiability Problem*, pages 127–136, 1996. Also available as Technical Report No. 96-230, Universität zu Köln, Germany.

[Sch99] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 410–414. IEEE Computer Society Press, October 1999.

[Sch05] U. Schöning. Algorithmics in exponential time. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science*, pages 36–43. Springer-Verlag *Lecture Notes in Computer Science #3404*, 2005.

[Wan97a] J. Wang. Average-case computational complexity theory. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 295–328. Springer-Verlag, 1997.

[Wan97b] J. Wang. Average-case intractable NP problems. In D. Du and K. Ko, editors, *Advances in Languages, Algorithms, and Complexity*, pages 313–378. Kluwer Academic Publishers, 1997.

[Wil84] H. Wilf. Backtrack: An $\mathcal{O}(1)$ expected time algorithm for the graph coloring problem. *Information Processing Letters*, 18(3):119–121, 1984.

[Woe03] G. Woeginger. Exact algorithms for NP-hard problems. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *Combinatorical Optimization: "Eureka, you shrink!"*, pages 185–207. Springer-Verlag *Lecture Notes in Computer Science #2570*, 2003.

[Yam05] M. Yamamoto. An improved $\tilde{\mathcal{O}}(1.234^m)$-time deterministic algorithm for SAT. In *Proceedings of the 16th International Symposium on Algorithms and Computation*, pages 644–653. Springer-Verlag *Lecture Notes in Computer Science #3827*, 2005.