Journal on Satisfiability, Boolean Modeling and Computation 2 (2006) 27-46

Hard Satisfiable Clause Sets for Benchmarking Equivalence Reasoning Techniques

Harri Haanpää Harri.Haanpaa@tkk.fi
Matti Järvisalo Matti.Jarvisalo@tkk.fi
Petteri Kaski Petteri.Kaski@tkk.fi
Ilkka Niemelä Ilkka.Niemela@tkk.fi
Laboratory for Theoretical Computer Science,
Helsinki University of Technology,
P.O. Box 5400, FI-02015 TKK, Finland

Abstract

A family of satisfiable benchmark instances in conjunctive normal form is introduced. The instances are constructed by transforming a random regular graph into a system of linear equations followed by clausification. Schemes for introducing nonlinearity to the instances are developed, making the instances suitable for benchmarking solvers with equivalence reasoning techniques. An extensive experimental evaluation shows that state-of-the-art solvers scale exponentially in the instance size. Compared with other well-known families of satisfiable benchmark instances, the present instances are among the hardest.

Keywords: benchmarking, Boolean satisfiability, equivalence reasoning, satisfiable instances

Submitted October 2005; revised December 2005; published

1. Introduction

During the last ten years there has been continuous interest in developing increasingly robust and efficient solution techniques for the propositional satisfiability problem (SAT). This research is largely motivated by the emergence of SAT-based techniques as a powerful alternative to more domain-specific techniques in many applications [11, 17, 32, 35].

In this paper we are interested in constructing benchmarks useful for developing state-of-the-art clausal SAT solvers. In particular, the goal is to develop families of satisfiable benchmarks in conjunctive normal form (CNF) that are *empirically hard*. Ideally, this means families of benchmarks for which the running time of state-of-the-art SAT solvers (i) scales exponentially in the size of the instance and (ii) is substantial even for instances of modest size. In this work we focus on developing empirically hard benchmarks for two of the most successful classes of SAT solvers, i.e., those based on Davis-Putnam-Loveland-Logemann (DPLL) type algorithms and local search. From the perspective of solver development, benchmarks that are empirically hard for a particular class of SAT solvers reveal the practical gains of introducing new techniques (such as equivalence reasoning) into the solvers.

Benchmark sets arising from industrial applications constitute perhaps the most relevant test cases from a practical perspective. However, they have some limitations especially in early stages of solver development. Industrial benchmarks typically come only as individual instances that are quite large, which makes them difficult to use in testing new algorithmic ideas, as large instances require substantial effort on optimizing key routines in the solver. Moreover, benchmarks consisting of a limited number of individual instances are not well suited for measuring improvement and scalability.

There are a number of properties that can be considered desirable for a family of benchmarks, especially with solver development in mind. The family should have control parameters—such as size—that provide control over the difficulty of a test case. Test cases with the same parameter values should have similar computational characteristics (are roughly equally challenging) and already fairly small test cases should be empirically hard. Test cases should have structure that could be exploited by an advanced solver and it should be easy to generate large numbers of guaranteed satisfiable (unsatisfiable) test cases of the same size.

A number of crafted families of unsatisfiable instances [16, 23, 50, 51] are known to be hard not only empirically but also in a stronger sense. In particular, it has been shown that minimal resolution proofs for various CNF families are exponential. Such results imply that all SAT solvers based on proof systems that can be polynomially simulated by resolution—including DPLL-based solvers [9]—scale exponentially on the considered CNF families.

Here we focus on hard *satisfiable* instances, which have turned out to be more difficult to construct. In many applications the instances of interest are in fact satisfiable and the key task of a SAT solver is to find a satisfying truth assignment [11, 17, 32, 35]. Hence, good performance for satisfiable instances is very important in practice. Satisfiable test cases are also useful in comparing different heuristics in their ability to guide the search towards a satisfying truth assignment. Moreover, only satisfiable instances are relevant for benchmarking incomplete (local) search methods.

Among the well-known sources of empirically hard instances are the random k-SAT [18] model and its restrictions such as regular random k-SAT [14]. However, the satisfiability of random k-SAT instances cannot be determined efficiently beforehand. This problem has been addressed by "hiding" solutions by generating only clauses that are satisfied by truth assignments selected beforehand [2, 7, 29]. More structured satisfiable benchmark instances have been developed based on, for example, quasigroup completion [1, 31]. However, a large number of variables is required in the clausal encoding of quasigroup completion to obtain a challenging instance for state-of-the-art SAT solvers.

Although systems of linear equations modulo 2 are polynomial-time solvable by Gaussian elimination, linear equation systems presented in CNF (XORSAT) are a well-known source of empirically hard instances for DPLL solvers. For example, in the area of circuit verification and logical cryptanalysis there are problems involving linear substructure (XOR equations) that are very challenging for CNF-based solvers [8]. The random k-XORSAT model [45] even exhibits a phase transition phenomenon similar to that of random k-SAT. Starting from the seminal work of Tseitin [50], ways of transforming different types of graphs into linear equation systems modulo 2 have been considered for constructing notably hard CNF families. Such models enable the generation of instances with predefined satisfiability. Many of the proposed satisfiable XORSAT families are motivated by spin glass models from statistical physics [22, 28, 41]. Recently, satisfiable XORSAT has also been examined in a complexity theoretic setting [3].

In this paper we develop a satisfiable CNF family—(*random*) *regular XORSAT*—by transforming *random regular graphs* into systems of linear equations modulo 2 presented in clausal form. The idea behind constructing empirically hard satisfiable XORSAT instances for DPLL-based solvers is to limit the effectiveness of Boolean constraint propagation as much as possible. In this respect, the underlying graph should be "highly connected" and, on the other hand, the number of occurrences of each variable should be as small as possible. The novelty here is that we employ random regular graphs, i.e., random graphs with each vertex having the same number of neighbors, to force these properties. Due to the simplicity of the model, it is easy to generate large numbers of instances of the same size. Moreover, we develop techniques for introducing nonlinearity (other Boolean connectives than XOR) into the equation system to make the benchmarks challenging also for clausal solvers equipped with equivalence reasoning techniques, i.e., special methods for solving linear equations presented in clausal form.

We perform extensive experimental evaluation of regular XORSAT with the following main conclusions.

- Both DPLL based and local search state-of-the-art SAT solvers scale exponentially on regular XORSAT. Furthermore, all instances of the same size are roughly equally challenging.
- By introducing nonlinearity into regular XORSAT, also DPLL solvers with equivalence reasoning techniques scale exponentially. However, we observe significant differences in the effectiveness of different equivalence reasoning techniques.
- Compared with several other families of satisfiable benchmarks, regular XORSAT is among the hardest with respect to the number of variables. As suggested by the results of the SAT Competition 2005 (see `http://www.satcompetition.org/`), already small instances of regular XORSAT are very hard to solve.

The paper is organized as follows. Preliminaries are presented in Section 2. Section 3 introduces regular XORSAT and offers a hypothesis to explain the empirical hardness of the instances. Experimental evaluation of regular XORSAT against other known families of hard satisfiable instances is presented in Section 4.

2. Preliminaries

Let X be a set of Boolean variables. Associated with every variable $x \in X$ there are two *literals*, the positive literal, denoted by x, and the negative literal, denoted by $\bar{x}$. A *clause* of length k (a k-clause) is a disjunction of k distinct literals. A propositional formula in *conjunctive normal form* (a CNF formula) is a conjunction of clauses. We adopt the standard convention of viewing a clause as a finite set of literals and a CNF formula as a finite set of clauses. A *truth assignment* τ associates a truth value $\tau(x) \in \{0, 1\}$ with each variable $x \in X$. A truth assignment *satisfies* a CNF formula if it satisfies every clause in it. A clause is satisfied if it contains at least one satisfied literal, where a literal x (respectively, $\bar{x}$) is satisfied if $\tau(x) = 1$ (respectively, $\tau(x) = 0$).

The *satisfiability problem* (SAT) is to decide whether a given CNF formula (the "problem instance") admits a satisfying truth assignment. If every clause in the instance has length k, then we speak of the k-SAT problem.

A *graph* is a pair $G = (V, E)$, where V is a finite set and E is a set of 2-element subsets of V. The elements of V are called *vertices* and the elements of E *edges*. Two vertices $u, v \in V$ are *adjacent* if $\{u, v\} \in E$. A vertex $u \in V$ is *incident* to an edge $e \in E$ if $u \in e$. The *degree* d of a vertex is the number of vertices adjacent to it. A graph is *d-regular* if all of its vertices have degree d. A graph is *bipartite* if there exist $X, Y \subseteq V$ such that $X \cup Y = V$, $X \cap Y = \emptyset$, and every edge is incident to one vertex in X and one vertex in Y. Such a pair (X, Y) is a *bipartition* of the graph.

3. Regular XORSAT

In this section we describe the proposed family of hard satisfiable benchmarks which we call regular XORSAT. Roughly, the generation procedure is as follows. First, a 3-regular constraint graph is selected uniformly at random. Then, a system of linear equations modulo 2 based on the graph is constructed. Finally, the actual regular XORSAT instance is obtained by transforming the equations into an equivalent set of clauses. As an optional postprocessing step, various degrees of nonlinearity may be introduced into the clause set.

We now proceed to describe and analyze regular XORSAT in more detail.

3.1 The Basic Construction

Let n be the number of variables in the regular XORSAT instance to be constructed. Let $X = \{x_0, x_1, \ldots, x_{n-1}\}$ be an associated set of n Boolean variables and let $Y = \{y_0, y_1, \ldots, y_{n-1}\}$ be a set of n elements, each corresponding to an equation in a system of n equations over X. A *constraint graph* $G = (V, E)$ with bipartition (X, Y) characterizes the occurrences of the variables in the equations, that is, $\{x_j, y_i\}$ is an edge of G if and only if the variable $x_j \in X$ occurs in the equation $y_i \in Y$. The foundation of our construction lies in selecting a constraint graph G uniformly at random from the set of all 3-regular graphs with bipartition (X, Y). As a running example, consider the graph in Figure 1.

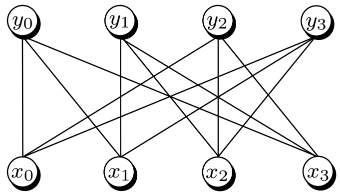

Figure 1. A 3-regular constraint graph

Once a constraint graph G has been selected, construct a system of linear equations based on G as follows. Let $A = (a_{ij})$ be the $n \times n$ matrix whose entries are defined for all $i, j = 0, 1, \ldots, n-1$ by

$$a_{ij} = \begin{cases} 1 & \text{if } \{x_j, y_i\} \in E, \\ 0 & \text{if } \{x_j, y_i\} \notin E. \end{cases}$$

Select uniformly at random a $\vec{z} \in \{0,1\}^n$ and let $\vec{b} \in \{0,1\}^n$ so that $\vec{b} \equiv A\vec{z} \pmod 2$. The system of linear equations is now $A\vec{x} \equiv \vec{b} \pmod 2$, where $\vec{x} = (x_0, x_1, \ldots, x_{n-1})$ is a column vector of variables. Note that by construction $A\vec{z} \equiv \vec{b} \pmod 2$, so the system always has at least one solution—if a unique solution is required, then the matrix A must be invertible modulo 2. As an example, from the constraint graph in Figure 1 we obtain the matrix

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

For a randomly chosen vector

$$\vec{z} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad \text{we arrive at} \quad \vec{b} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

and the equation system

$$\begin{cases} x_0 + x_1 + x_3 & = & 0 \\ x_1 + x_2 + x_3 & = & 0 \\ x_0 + x_2 + x_3 & = & 1 \\ x_0 + x_1 + x_2 & = & 0 \end{cases} \quad \pmod 2.$$

Finally, transform the system $A\vec{x} \equiv \vec{b} \pmod 2$ into a CNF formula by introducing for every equation $x_{j_1} + x_{j_2} + x_{j_3} \equiv b_i \pmod 2$ a set of four clauses that forbid the combinations of truth values that violate the equation. For example, the equation $x_0 + x_1 + x_2 \equiv 0 \pmod 2$ transforms into the clauses $\{\bar{x}_0, \bar{x}_1, \bar{x}_2\}$, $\{\bar{x}_0, x_1, x_2\}$, $\{x_0, \bar{x}_1, x_2\}$, and $\{x_0, x_1, \bar{x}_2\}$. For the running example, the resulting regular XORSAT instance is

$$\begin{aligned} \{ & \{\bar{x}_0, \bar{x}_1, \bar{x}_3\}, \{\bar{x}_0, x_1, x_3\}, \{x_0, \bar{x}_1, x_3\}, \{x_0, x_1, \bar{x}_3\}, \\ & \{\bar{x}_1, \bar{x}_2, \bar{x}_3\}, \{\bar{x}_1, x_2, x_3\}, \{x_1, \bar{x}_2, x_3\}, \{x_1, x_2, \bar{x}_3\}, \\ & \{\bar{x}_0, \bar{x}_2, x_3\}, \{\bar{x}_0, x_2, \bar{x}_3\}, \{x_0, \bar{x}_2, \bar{x}_3\}, \{x_0, x_2, x_3\}, \\ & \{\bar{x}_0, \bar{x}_1, \bar{x}_2\}, \{\bar{x}_0, x_1, x_2\}, \{x_0, \bar{x}_1, x_2\}, \{x_0, x_1, \bar{x}_2\}\}. \end{aligned}$$

3.2 Motivation for the Regular XORSAT Model

The idea of transforming a graph into a system of linear equations followed by clausification dates back at least to Tseitin's seminal work on resolution proof complexity [50]. Compared with Tseitin's construction, we employ a somewhat different transformation from a graph into equations, and focus on satisfiable instances rather than the unsatisfiable instances employed in proof complexity theory [10, 16, 51]. Recently, also satisfiable instances based on clausification of linear equations have been examined in a complexity theoretic setting [3].

The two existing benchmark families most resembling ours are the random 3-XORSAT instances [45] and the 3-XORSAT family described by Jia et al. [28] motivated by spin glass models from statistical physics. Comparing regular XORSAT and random 3-XORSAT (with forced satisfiability), our construction uses a random regular constraint graph whereas in random 3-XORSAT the constraint graph is formed by associating independently and uniformly a set of three variables with each of the m equations, m being an additional parameter. Thus, given n variables our construction produces exactly n equations with exactly three occurrences of each variable, whereas in random 3-XORSAT the number of occurrences of a variable is a binomially distributed random variable with expectation $3m/n$. Comparing regular XORSAT and the spin glass 3-XORSAT family described by Jia et al. [28], our construction selects the regular constraint graph uniformly at random whereas Jia et al. associate with each (square) n a fixed regular constraint graph derived from a $\sqrt{n} \times \sqrt{n}$ rhombic lattice with cyclic boundary.

Given this resemblance to existing benchmarks, it is not immediate why combining regularity and random selection should yield results any different compared with existing benchmarks. For example, the clauses-to-variables ratio commonly used to predict satisfiability and computational difficulty in the context of random 3-SAT [18] and random 3-XORSAT [45] is equal to 4 for both regular XORSAT and the spin glass 3-XORSAT instances: each linear equation with 3 variables clausifies to 4 clauses. Nevertheless the experiments documented in Section 4 show that regular XORSAT instances exhibit more rapid exponential scaling for state-of-the-art SAT solvers than the aforementioned benchmarks. In what follows we motivate this observed difference by explaining why a DPLL-type algorithm without learning could find regular XORSAT instances challenging.

A DPLL-type algorithm without conflict learning performs a systematic search for a satisfying truth assignment for the given set of input clauses. In an abstract setting, it can be seen as a depth-first traversal of a search tree whose every internal node is associated with a *split variable* x_j, and the two child nodes of such a node correspond to setting $\tau(x_j) = 0$ and $\tau(x_j) = 1$, respectively. Following each split, Boolean constraint propagation (BCP) is applied to determine the values of additional variables based on the values assigned to the split variables in the path from the current node to the root node of the tree. This can lead to a satisfying truth assignment in which case the search can be stopped. However, if BCP leads to a conflicting assignment for a variable, the algorithm backtracks to the nearest split variable for which the other branch has not yet been explored and continues from there.

In what follows we consider the case where a DPLL type algorithm is working on a set of clauses representing a system of linear equations. In this discussion it is convenient to look at BCP on the level of linear equations rather than clauses. On this level BCP can be seen as a procedure that computes the closure of the simple rule of finding an equation with only one variable whose truth value has not been assigned and setting the truth value as dictated by the equation. This connection is quite straightforward to establish because the steps in BCP have clear counterparts on the level linear equations. On the level of clauses, BCP can be seen as the application of the following rule until no new truth value assignments can be derived: simplify the clause set given the current assignment of truth values and then extend the assignment for the 1-clauses in the simplified clause set by setting $\tau(x) = 1$ (respectively, $\tau(x) = 0$) if the clause $\{x\}$ ($\{\bar{x}\}$) appears in the set. Indeed, considering a set of clauses originating from linear equations, these steps can easily be traced back to the linear equations. The operation of simplifying a set of clauses when setting $\tau(x_i) = 1$ ($\tau(x_i) = 0$) means on the level of clauses that we remove all clauses containing the literal x_i ($\bar{x}_i$) and all literals $\bar{x}_i$ (x_i) from the remaining clauses—the clauses that remain are exactly the clausification of the linear equations with x_i set to 1 (0). Similarly, a 1-clause corresponds to an equation in which the truth value of all but one variable has been assigned.

To generate a small instance based on linear equations that is empirically hard for DPLL-type algorithms, essentially three properties are intuitively desirable.

(i) To keep the size of the resulting set of clauses small (without introducing additional variables), each equation should contain as few variables as possible, 3 variables being the smallest nontrivial number.

(ii) Each variable should occur in as few equations as possible to limit the implications of setting the truth value of a variable. If instances with a unique solution are desired, n equations are required for n variables, and thus each variable must occur on the average in at least 3 equations. A 3-regular constraint graph forces each variable to occur in exactly 3 equations.

(iii) The equation system should be "highly connected" to limit the effectiveness of BCP.

We proceed to discuss connectedness in more detail. Suggested in particular by the proofs of hardness in [3, 10, 51] (see also [9]), the connectedness of an equation system can be measured by the expansion properties of the underlying constraint graph. Here we focus on edge expansion. For a graph $G = (V, E)$ and a set $U \subseteq V$ of vertices, let $\partial_G U$ be the set of all edges in G that are incident with exactly one vertex in U. Call $\partial_G U$ the *boundary* of U. The *expansion coefficient* of G—alternatively, the *isoperimetric number* of G—is defined by

$$h(G) = \min \left\{ \frac{|\partial_G U|}{|U|} \ : \ U \subseteq V, \ 1 \leq |U| \leq \frac{|V|}{2} \right\}.$$

To provide an intuition why expansion is relevant in limiting the effectiveness of BCP, let us derive an upper bound on the performance of BCP in terms of the expansion coefficient $h(G)$ of the constraint graph G underlying a regular XORSAT instance. Our interest is to bound the number of determined variables based on the number of split variables and the expansion coefficient.

In the present context of linear equations with exactly 3 variables each, BCP can be seen as the application of the following rule until no more variables become determined: letting $S \subseteq X$ be the set of variables whose value has been determined so far, if the instance contains an equation with 2 variables in S and 1 variable $x_j \in X \setminus S$, then we can insert x_j into S. We denote by $\bar{S}$ the closure of S with respect to BCP. In each node of the DPLL search tree, the set of determined variables can be seen as the BCP closure of the set of split variables in the path from the current node to the root node. Recall that we write n for the number of variables.

Theorem 1. *If* $|S| \leq (n-2)h(G)/3$, *then* $|\bar{S}| \leq (3/(2h(G)) + 1/2)|S|$.

Proof. Initially, let $U = S$. We trace the operation of BCP in steps by enlarging the set U. Each application of the BCP rule can be viewed in terms of the constraint graph G and the set U as the following operation: if $x_{j_1}, x_{j_2}, x_{j_3} \in X$ are the 3 vertices adjacent to $y \in Y$, and it holds that $x_{j_1}, x_{j_2} \in U$ and $x_{j_3} \in X \setminus U$, then the value of x_{j_3} becomes determined through BCP. When this happens, we enlarge U by inserting both x_{j_3} and y into U. Now observe the following invariant: the size of U grows by 2 as the BCP rule is applied, but the boundary $\partial_G U$ either stays constant in size or shrinks. We apply this invariant to bound the size of the BCP closure. Initially we have $|\partial_G U| = 3|S|$. Thus, by the invariant and the definition of the expansion coefficient $h(G)$, we have (a) $|U| > n$ or (b) $h(G) \leq |\partial_G U|/|U| \leq 3|S|/|U|$ at all times during BCP. Initially (a) is false, so (b) is true. Because of the assumption $|S| \leq (n-2)h(G)/3$, by (b) we have $|U| \leq 3|S|/h(G) \leq n-2$

at all times unless (a) becomes true. Every step of BCP enlarges U by 2, so (a) never becomes true and (b) is true at all times. In particular, when BCP terminates, we have $|\bar{S}| = (|U| - |S|)/2 + |S| \leq (3/(2h(G)) + 1/2)|S|$. $\square$

Assuming that $h(G)$ has a constant nonzero lower bound as the number of variables n increases, the previous theorem shows that the number of variables whose value is determined by BCP is linearly bounded by the number of split variables. Thus, assuming that conflicts are infrequent until the value of a large number of variables is determined, the previous theorem shows that many split variables are required and thus the DPLL search tree will be large. Thus, hypothetically, the larger the expansion coefficient $h(G)$, the larger the search tree.

To motivate our choice of random regular graphs in this light, we first observe that computing $h(G)$ for a given graph G is NP-hard [12]. However, $h(G)$ can be bounded for a d-regular graph G in terms of $\lambda_2(G)$, the second largest eigenvalue of an adjacency matrix of G, as follows [4, 5, 48]:

$$\frac{d - \lambda_2(G)}{2} \leq h(G) \leq \sqrt{2d(d - \lambda_2(G))}. \tag{1}$$

The construction of explicit infinite families of d-regular graphs with a constant nonzero lower bound on $h(G)$ is a nontrivial task; see [44] and the references therein for an account of known explicit constructions. For example, the family of 3-regular constraint graphs underlying the spin glass XORSAT instances in [28] is *not* expanding in this sense—it can be checked that the expansion coefficient has an $O(1/\sqrt{n})$ upper bound. Fortunately, most d-regular graphs have good expansion properties [13, 33], so perhaps the easiest and most versatile way to obtain a d-regular graph with good expansion properties is to select one uniformly at random. For our present purposes we require a 3-regular graph admitting a fixed bipartition; also with this restriction it is possible to prove that most graphs admit a constant nonzero lower bound on $h(G)$. In practice experimentation suggests that the standard uniform sampling procedure we use (see Section 4.1) produces graphs with $\lambda_2(G)$ close to $2\sqrt{2} \approx 2.8284$, which is the asymptotic optimum in terms of (1) for $\lambda_2(G)$ on 3-regular graphs [4, 39].

3.3 Schemes for Introducing Nonlinearity

A system of linear equations modulo 2 cannot in itself be considered hard; both the existence and nonexistence of a solution can easily be determined by Gaussian elimination. However, DPLL itself does not include any special techniques for equivalence reasoning. As linear substructures often occur in real-world application domains of DPLL solvers (such as hardware verification), the gains from introducing equivalence reasoning into DPLL solvers seem evident. Indeed, equivalence reasoning techniques are a recent development in DPLL-based SAT solvers [6, 8, 25, 38, 43, 52]. To facilitate benchmarking of equivalence reasoning techniques, we propose the following schemes for introducing nonlinearity into regular XOR-SAT.

Naive Scheme. Introduce three new variables x, y, z, and insert the literal x into each original clause. Additionally, add 7 clauses that force x to 0 and y, z into unique truth values.

Covering Scheme. Select a minimal set of variables such that every clause contains at least one selected variable. For each selected variable, x, introduce a new variable, y, and then substitute each occurrence of x (respectively, $\bar{x}$) in the clauses with $x \wedge y$ (respectively, $\overline{x \wedge y} \equiv \bar{x} \vee \bar{y}$). After all the substitutions have been performed, expand any conjuncts inside disjuncts to obtain a set of clauses.

The naive scheme is intended for benchmarking preprocessors with respect to their ability to detect the clausal representation of a set of linear equations that is conditional on a single variable x. The covering scheme is designed for benchmarking dynamic equivalence reasoning techniques that are applied during search. Ideally, a solver should be able detect and exploit the underlying linear substructure that is revealed when variables are assigned truth values during search. Note that the number and the lengths of clauses are somewhat affected by the schemes.

These two basic schemes can be extended as follows.

k-Nonlinear Covering Scheme. As an extension of the covering scheme, instead of introducing one new variable y for each selected variable x, introduce k new variables $y_1, \ldots, y_k$ for each such x. Substitute each occurrence of x (respectively, $\bar{x}$) in the clauses with $x \wedge y_1 \wedge \cdots \wedge y_k$ (respectively, $\overline{x \wedge y_1 \wedge \cdots \wedge y_k} \equiv \bar{x} \vee \bar{y}_1 \vee \cdots \vee \bar{y}_k$), and expand any resulting conjuncts to obtain clauses.

p-Covering Scheme. Select a minimal set of variables such that $p\%$ of the clauses contain at least one selected variable. Apply the k-nonlinear covering scheme on these selected variables only.

p-Mixed Covering/Naive Scheme. As in p-covering scheme, but additionally apply the naive scheme for the remaining $(100 - p)\%$ clauses not containing any selected variables.

Note that these schemes can be adopted to other SAT benchmark classes based on linear equations.

4. Experiments

In this section we report on experimental evaluation comparing regular XORSAT instances with other well known families of hard satisfiable instances using both DPLL based and local search state-of-the-art SAT solvers.

We submitted a benchmark collection based on regular XORSAT to the SAT Competition 2005 (see `crafted/jarvisalo05`). To the best of our knowledge, our benchmarks provided the smallest guaranteed satisfiable instances that were not solved by any solver in the second stage of the competition. The smallest such instance was based on a basic regular XORSAT instance with 230 variables, which with the covering scheme had 322 variables. There were smaller unsolved instances in the competition, in particular, those generated with OKgenerator [34] but it is unclear whether they are satisfiable because the generation method does not guarantee this.

Before discussing the experimental results, we now describe the implementation techniques used for generating regular XORSAT instances.

4.1 The Generator

We generate uniformly at random 3-regular graphs on $2n$ vertices admitting a fixed bipartition using a restricted version of the pairing model [53]. For a given $n \geq 3$, let $X = \{x_0, x_1, \ldots, x_{n-1}\}$ and $Y = \{y_0, y_1, \ldots, y_{n-1}\}$ define the fixed bipartition (X, Y) and let $V = X \cup Y$ be the vertex set. Select a permutation $\pi : \{0, 1, \ldots, 3n-1\} \rightarrow \{0, 1, \ldots, 3n-1\}$ uniformly at random. For every $i = 0, 1, \ldots, 3n-1$, introduce the edge $\{x_{i \bmod n}, y_{\pi(i) \bmod n}\}$. If there are repeated edges, reject the result and try again with a new permutation. It follows from a result of O'Neil [42, Theorem 2.3] (see also [53]) that the probability of no repeated edges approaches $\exp(-2) \approx 0.1353$ as n goes to infinity. In practice, only a few retries are required.

From a generated constraint graph we construct the matrix A as in Section 3.1. If a unique satisfying truth assignment is desired, we use Gaussian elimination to filter out instances in which the resulting matrix A is not invertible modulo 2. In practice it appears that for $n \leq 1000$ around one fourth of the generated matrices A are invertible, and most of the remaining matrices have a kernel of dimension 1 or 2, implying that the resulting CNF formula has at most four satisfying truth assignments.

When applying the covering scheme, a greedy approximation algorithm [30] for the set cover problem is employed for finding an appropriate set of variables for substitution: each variable vertex in the constraint graph covers the adjacent vertices associated with linear equations.

The generator software and the instances submitted to the SAT Competition 2005 are available at `http://www.tcs.hut.fi/Software/`.

4.2 Hardness Comparison

We compare the empirical hardness of regular XORSAT to other known families for both DPLL and local search. The following benchmark families are considered:

- 3-regular, unique: regular XORSAT with exactly one satisfying truth assignment,
- 3-regular, nonunique: regular XORSAT with at least one satisfying truth assignment,
- random 3-XORSAT: satisfiable random 3-XORSAT at the phase-transition point $\alpha_x = 0.918$ [45],
- Jia et al's 3-XORSAT: Jia et al's generator [28] motivated by a spin glass model [41] on a rhombus with cyclic boundary conditions (satisfiable "spin glass formulas"),
- random 3-SAT: random 3-SAT at the phase transition point $\alpha_s = 4.27$ [18], and
- q-hidden: Jia et al's generator for "deceptive q-hidden" satisfiable 3-SAT formulas [29] at $q = 0.3$ and at the threshold $q = 0.618$ (q-hidden formulas).

In generating all XORSAT instances, linear equations are clausified exactly in the same way as in regular XORSAT. The random 3-SAT instances are generated with `makewff` [46] and the q-hidden instances with a generator obtained directly from the authors of [29]. In the case of random 3-SAT we do not filter out unsatisfiable instances; approximately 50% of the instances are unsatisfiable. We omit the hidden formulas reported in [2] and the `hgen2` generator [26] from consideration; it has been observed [29] that q-hidden formulas are more difficult than those in [2], while `hgen2` does not generate instances with fewer than 250 variables.

4.2.1 Complete Search

Considering DPLL-based solvers, we compare the hardness of regular XORSAT instances against random 3-XORSAT, Jia et al's 3-XORSAT, random 3-SAT, and q-hidden instances. The solvers used are `SatEliteGTI` (that is, the `MiniSat` solver [20, 21] with the `SatElite` preprocessor [19]), `zChaff` [40, 54], and `Satz` [36, 37]. This choice is motivated by

(i) `SatEliteGTI`'s award-winning performance in the SAT Competition 2005,

(ii) `zChaff`'s use of conflict-driven learning and its position as one of the most widely and successfully used solvers in recent years,

(iii) the fact that preliminary experiments on a wider array of solvers (namely, `COMPSat`, `HaifaSat`, `Jerusat`, `Satzoo`, and `Siege`) suggest that these DPLL solvers perform similarly to (or, sometimes, slightly worse than) `zChaff` on our instances, and

(iv) `Satz` being one of the most widely used solvers without conflict learning and substantially different from `zChaff`.

Depending on the solver, we plot the number of decisions or the number of branches as reported by the solver. In Figures 2, 3, and 4 we display for each number of variables the median number of decisions/branches over 15 instances on a base-10 logarithmic scale. Among the instance families, our generator gives the hardest instances for `SatEliteGTI`, `zChaff` and `Satz`.

4.2.2 Local Search

We experiment with the local search solvers `WalkSAT` [46, 47] and `AdaptiveNovelty+`[27, 49]. While `SP` (survey propagation) [15] is extremely efficient in solving random k-SAT formulas, it has been observed to exhibit very poor performance on XORSAT [28]; thus we do not consider `SP` here.

We compare regular XORSAT to random 3-XORSAT, Jia et al's 3-XORSAT, and q-hidden instances. In Figures 5 and 6 we plot for each number of variables the base-10 logarithm of the median number of flips over 49 trials. These local search solvers do not scale as well as DPLL solvers and, hence, the size of the instances studied is smaller than that for DPLL solvers. Interestingly, while decreasing the value of q makes q-hidden instances harder for local search [29], we do not observe the same effect for DPLL solvers. Regular XORSAT appears to be the hardest of the considered families for local search, too.

4.3 Equivalence Reasoning: A Solver Comparison

To investigate how well current equivalence reasoning techniques manage when nonlinearity is introduced, we compare the behavior of DPLL solvers on our basic regular XORSAT instances, on instances with the naive scheme added, and on instances with the covering scheme. Note that (i) with the naive scheme, the effective number of variables is 7 more than the original number of variables, while (ii) with the covering scheme, the number of added variables depends on the set cover. Applying the covering scheme, the number of

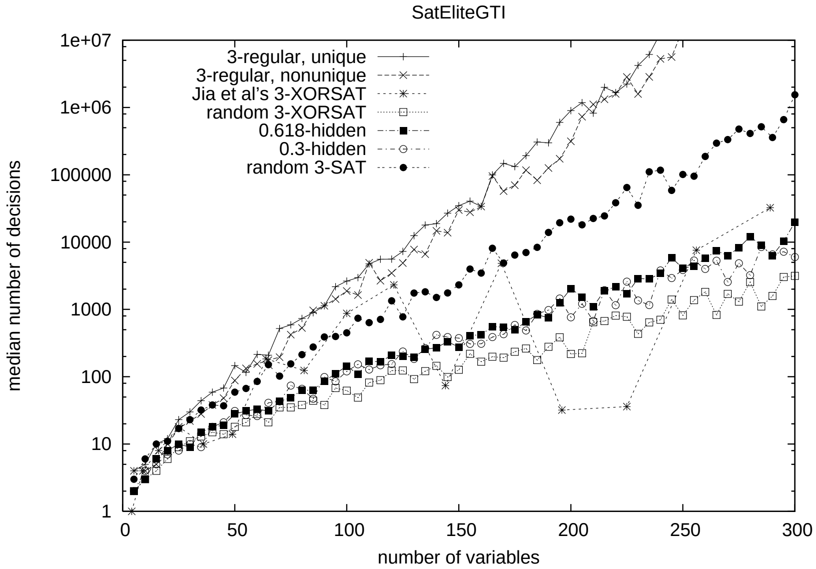

Figure 2. Hard instance families for DPLL-based solvers: median of number of decisions as reported by `SatEliteGTI`.

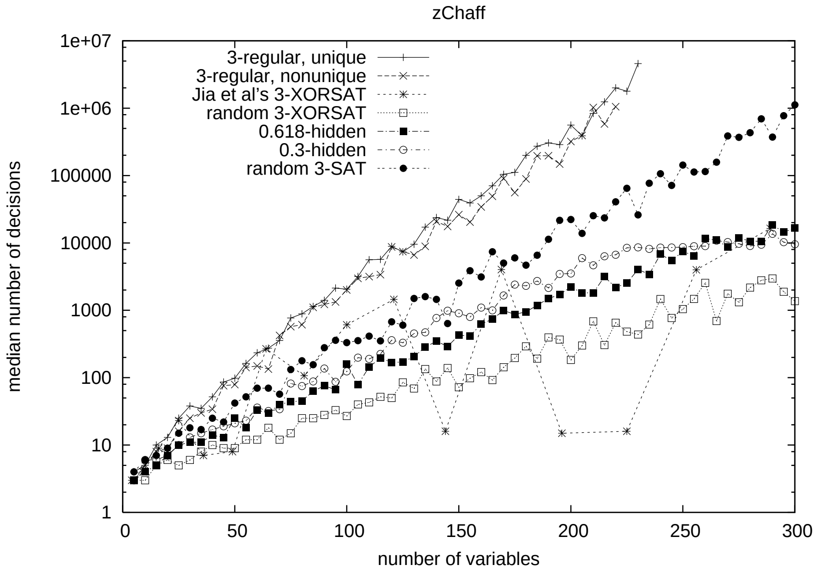

Figure 3. Hard instance families for DPLL-based solvers: median of number of decisions as reported `zChaff`.

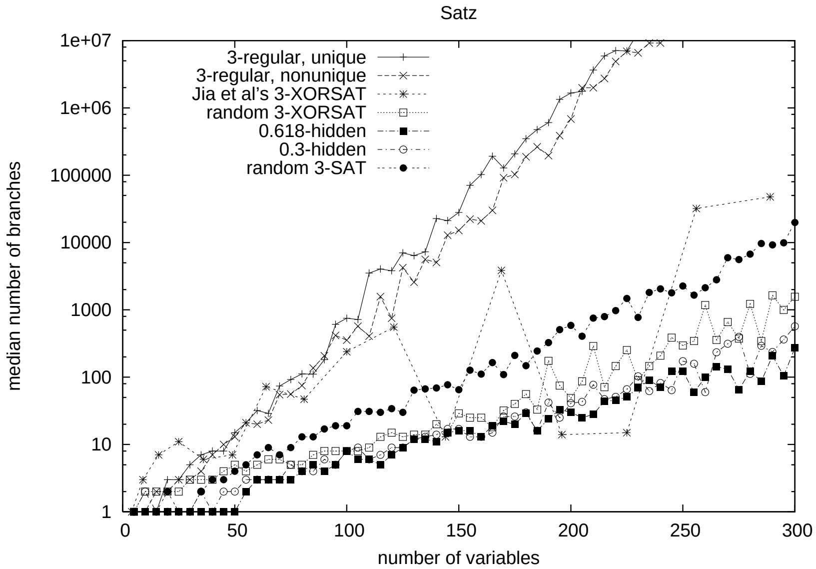

Figure 4. Hard instance families for DPLL-based solvers: median of number of branches as reported by `Satz`.

variables increases 35% to 50% in the instances used in the experiments. For instances with 200 or more variables prior to applying the covering scheme, this percentage is close to 40. For example, in the instances with 250 variables prior to applying the covering scheme, the maximum number of variables is 357 with the covering scheme. In other words, by applying the covering scheme we do not drastically increase the number of variables. The number and the lengths of clauses are also somewhat increased by the schemes. However, this is unproblematic in practise because current state-of-the-art solvers should be designed to handle variable length clauses which are very common in practical applications, too.

In addition to `SatEliteGTI`, `zChaff` and `Satz`, we use the solvers `march_dl` [24, 25] and `EqSatz` [36, 38], both of which incorporate equivalence reasoning techniques. The results are shown in Figure 7. For each number of variables, we plot both the median running time (right) and the number of decisions (left) over 15 instances on a base-10 logarithmic scale.

The experiments illustrate the following differences in `march_dl` and `EqSatz`. Figure 7 (top) indicates that the equivalence reasoning techniques in `march_dl` are more complete in the sense that `march_dl` can solve purely linear problems very efficiently while `EqSatz` is able to do this only up to a certain size after which it scales exponentially. The results in Figure 7 (middle) imply that the techniques in `march_dl` are somewhat fragile: efficiency is lost when a minor amount of nonlinearity is introduced. On the other hand, the techniques in `EqSatz` are more robust as `EqSatz` maintains almost the same efficiency in the new setting. Figure 7 (bottom) demonstrates that equivalence reasoning techniques in `EqSatz` lose a substantial amount of efficiency when more nonlinearity is introduced. However, we observe that `march_dl` is competitive with `SatEliteGTI` on these benchmarks.

Another interesting observation concerns the ratio between the number of decisions and the running times in solvers using learning techniques, `SatEliteGTI` and `zChaff`. For ex-

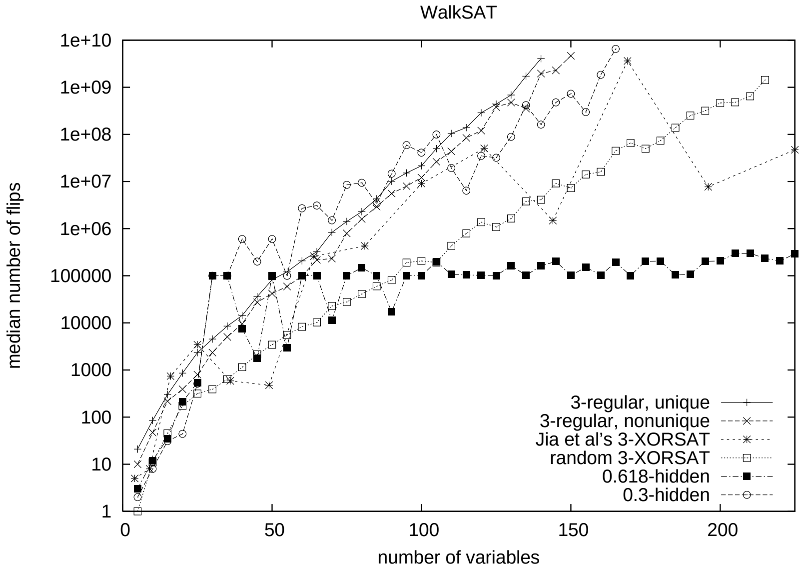

Figure 5. Hard satisfiable instance families for local search: median of number of flips as reported by `WalkSAT`.

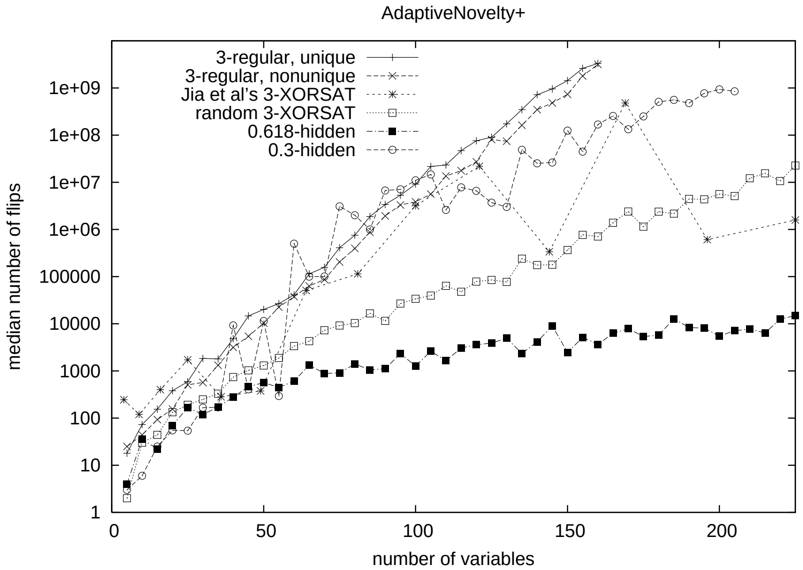

Figure 6. Hard satisfiable instance families for local search: median of number of flips as reported by `AdaptiveNovelty+`.

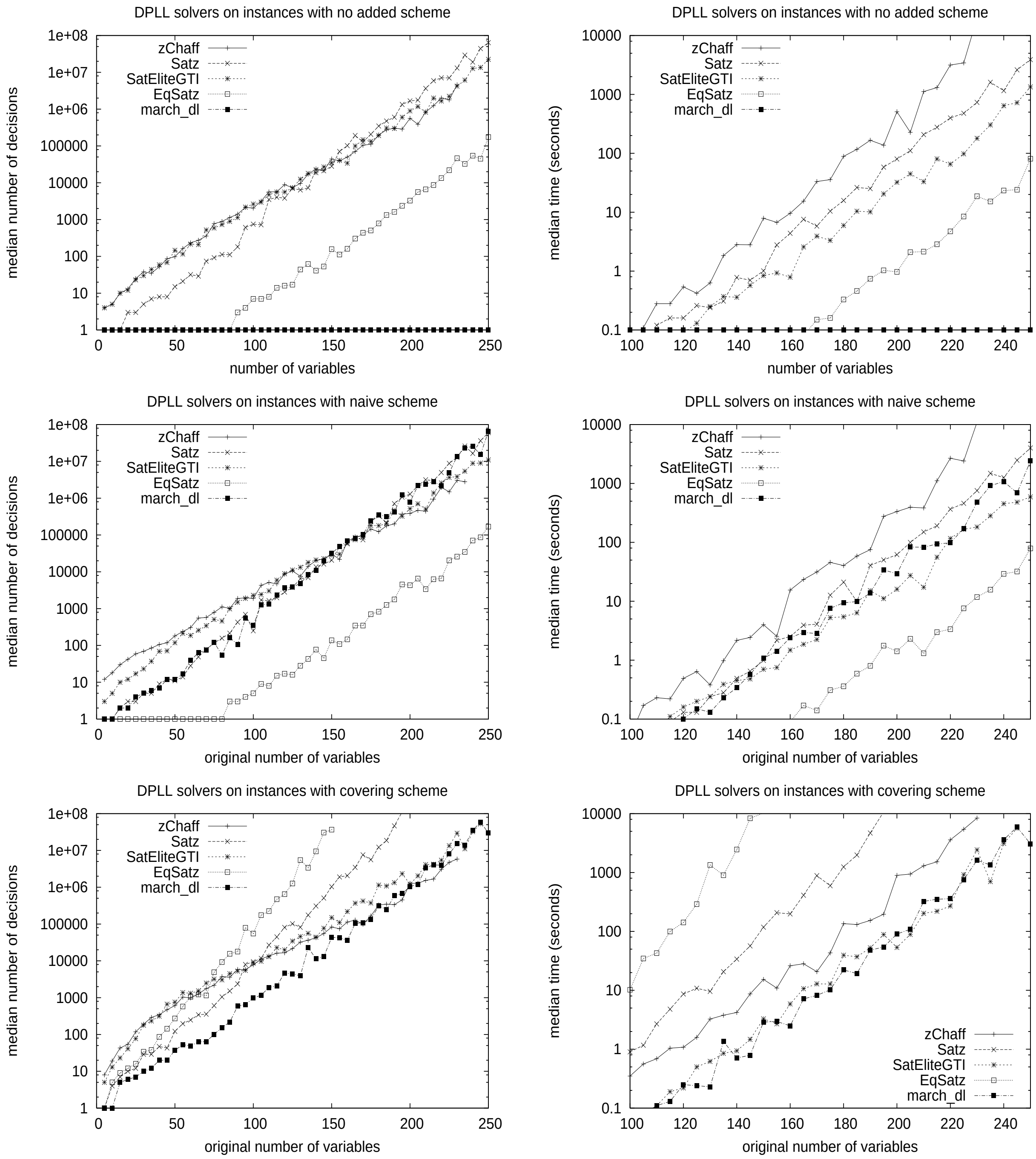

Figure 7. Comparison of different solvers against added nonlinearity: median number of decision (left) and median running times (right), with no added nonlinearity (top), with the naive scheme (middle), and with the covering scheme (bottom).

ample, Figure 7 (top and middle) shows that while the number of decisions for `SatEliteGTI` and `zChaff` are nearly equal, `SatEliteGTI` is an order of magnitude faster. As the main difference between `SatEliteGTI` and `zChaff` is the way in which clauses are learned and learned clauses are maintained, this result suggests that subtleties in these techniques are behind the substantial difference in the running times.

5. Conclusions

We propose a novel family of SAT instances—(random) regular XORSAT—derived from linear equations based on random regular graphs. Experimental evaluation shows that regular XORSAT instances appear to scale exponentially for both complete and incomplete state-of-the-art clausal SAT solvers. Compared with other well-known families of hard benchmark instances with guaranteed satisfiability, regular XORSAT instances are among the hardest. The proposed schemes for introducing nonlinearity make the instances especially suitable for benchmarking equivalence reasoning techniques.

To keep the instance size small in relation to the number of variables, in the present study we have considered only instances in which the underlying constraint graph is d-regular with $d = 3$. A topic for further study is to investigate instances with $d > 3$.

Here we have focused only on instances with guaranteed satisfiability. Although also unsatisfiable instances can be easily obtained by selecting a matrix A that is not invertible modulo 2 and choosing $\vec{b}$ outside the column span of A, we consider it likely that such unsatisfiable instances are easier to solve than, for example, the unsatisfiable instances obtained from Tseitin's construction [50, 51]. A detailed empirical comparison remains to be conducted.

From a theoretical perspective one topic for further research is to carry out a more rigorous analytical study of regular XORSAT, in particular in the context of local search. One way to motivate empirical hardness for local search is through the existence of many local minima surrounded by "potential barriers" that make escaping from such minima difficult [28]. We consider it possible that the expansion properties of the underlying constraint graph could be used to motivate hardness in this setting as well.

Our experiments suggest that there is room for further research in equivalence reasoning techniques. As structured real-world problem instances often contain linear substructure, we raise this as an important aspect of solver development.

Acknowledgments. The authors are grateful to the anonymous referees for their comments and suggestions which helped to improve the presentation of the paper. This research is supported in part by the Academy of Finland under grants 209300 (H.H.) and 211025 (M.J. and I.N.), Helsinki Graduate School in Computer Science and Engineering (M.J. and P.K.), and by grants from the Emil Aaltonen Foundation (M.J.) and the Nokia Foundation (P.K.).

References

[1] D. Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating satisfiable problem instances. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 256–261. AAAI Press, 2000.

[2] D. Achlioptas, H. Jia, and C. Moore. Hiding truth assignments: Two are better than one. *Journal of Artificial Intelligence Research*, 24:623–639, 2005.

[3] M. Alekhnovich, E.A. Hirsch, and D. Itsykson. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. In E. Giunchiglia and T. Walsh, editors, *SAT 2005, Satisfiability Research in the Year 2005*. Springer, 2006.

[4] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[5] N. Alon and V.D. Milman. λ_1, isoperimetric inequalities for graphs and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38(1):73–88, 1985.

[6] F. Bacchus. Enhancing Davis-Putnam with extended binary clause reasoning. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pages 613–619. AAAI Press, 2002.

[7] W. Barthel, A.K. Hartmann, F. Ricci-Tersenghi, M. Weight, and R. Zecchina. Hiding solutions in random satisfiability problems: A statistical mechanics approach. *Physical Review Letters*, 18:188701, 2002.

[8] P. Baumgartner and F. Massacci. The taming of the (X)OR. In *Computational Logic*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 508–522. Springer, 2000.

[9] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

[10] E. Ben-Sasson and A. Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001.

[11] A. Biere and W. Kunz. SAT and ATPG: Boolean engines for formal hardware verification. In *Proceedings of the 20th IEEE/ACM International Conference on Computer Aided Design*, pages 782–785. IEEE, 2002.

[12] M. Blum, R.M. Karp, O. Vornberger, C.H. Papadimitriou, and M. Yannakakis. The complexity of testing whether a graph is a superconcentrator. *Information Processing Letters*, 13(4-5):164–167, 1981.

[13] B. Bollobás. The isoperimetric number of random regular graphs. *European Journal of Combinatorics*, 9(3):241–244, 1988.

[14] Y. Boufkhad, O. Dubois, Y. Interian, and B. Selman. Regular random k-SAT: Properties of balanced formulas. In E. Giunchiglia and T. Walsh, editors, *SAT 2005, Satisfiability Research in the Year 2005*. Springer, 2006.

[15] A. Braunstein, M. Mezard, and R. Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms*, 27(2):201–226, 2005.

[16] V. Chvátal and E. Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, 1988.

[17] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[18] J.M. Crawford and L.D. Auton. Experimental results on the crossover point in random 3SAT. *Artificial Intelligence*, 81(1-2):31–57, 1996.

[19] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

[20] N. Eén and N. Sörensson. The miniSat page. `http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/`.

[21] N. Eén and N. Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[22] J.P. Garrahan and M.E.J. Newman. Glassiness and constrained dynamics of short-range non-disordered spin model. *Physical Review E*, 62:7670–7678, 2000.

[23] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, 1985.

[24] M. Heule and H. van Maaren. March_dl. `http://www.isa.ewi.tudelft.nl/sat/march_dl.htm`.

[25] M. Heule, J. van Zwieten, M. Dufour, and H. van Maaren. March_eq: Implementing additional reasoning into an efficient lookahead SAT solver. In *Theory and Applications of Satisfiability Testing, 7th International Conference*, volume 3542 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2005.

[26] E.A. Hirsch. hgen2 satisfiability instance generator. `http://logic.pdmi.ras.ru/~hirsch/sat.html`.

[27] H.H. Hoos. An adaptive noise mechanism for WalkSAT. In *18th National Conference on Artificial intelligence*, pages 655–660. AAAI Press, 2002.

[28] H. Jia, C. Moore, and B. Selman. From spin glasses to hard satisfiable formulas. In *Theory and Applications of Satisfiability Testing, 7th International Conference*, volume 3542 of *Lecture Notes in Computer Science*, pages 199–210. Springer, 2005.

[29] H. Jia, C. Moore, and D. Strain. Generating hard satisfiable formulas by hiding solutions deceptively. In *Proceedings of the 20h National Conference on Artificial Intelligence*, pages 384–389. AAAI Press, 2005.

[30] D.S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, pages 38–49. ACM Press, 1973.

[31] H. Kautz, Y. Ruan, D. Achlioptas, C. Gomes, B. Selman, and M. Stickel. Balance and filtering in structured satisfiable problems. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 351–358. Morgan Kaufmann, 2001.

[32] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 1194–1201. AAAI Press, 1996.

[33] A.V. Kostochka and L.S. Melnikov. On a lower bound for the isoperimetric number of cubic graphs. In *Probabilistic Methods in Discrete Mathematics*, volume 1 of *Progress in Pure and Applied Discrete Mathematics*, pages 251–265. VSP, 1993.

[34] O. Kullmann. OKgenerator, 2002. `http://cs-svr1.swan.ac.uk/~csoliver/OKgenerator.html`.

[35] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 11(1):6–22, 1992.

[36] C.M. Li. Satz v2.15 and eqsatz. `http://www.laria.u-picardie.fr/~cli/EnglishPage.html`.

[37] C.M. Li. A constraint-based approach to narrow search trees for satisfiability. *Information Processing Letters*, 71(2):75–80, 1999.

[38] C.M. Li. Equivalent literal propagation in Davis-Putnam procedure. *Discrete Applied Mathematics*, 130(2):251–276, 2003.

[39] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[40] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, pages 530–535. ACM, 2001.

[41] M.E.J. Newman and C. Moore. Glassy dynamics in an exactly solvable spin model. *Physical Review E*, 60:5068–5072, 1999.

[42] P.E. O'Neil. Asymptotics and random matrices with row-sum and column sum-restrictions. *Bulletin of the American Mathematical Society*, 75:1276–1282, 1969.

[43] R. Ostrowski, É. Grégoire, B. Mazure, and L. Sais. Recovering and exploiting structural knowledge from CNF formulas. In *Principles and Practice of Constraint Programming*, volume 2470 of *Lecture Notes in Computer Science*, pages 185–199. Springer, 2002.

[44] O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics, Second Series*, 155(1):157–187, 2002.

[45] F. Ricci-Tersenghi, M. Weight, and R. Zecchina. Simplest random K-satisfiability problem. *Physical Review E*, 63:026702, 2001.

[46] B. Selman and H. Kautz. Walksat v43. `http://www.cs.washington.edu/homes/kautz/walksat/`.

[47] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In D.S. Johnson and M.A. Trick, editors, *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532. American Mathematical Society, 1996.

[48] R.M. Tanner. Explicit concentrators from generalized N-gons. *SIAM Journal on Algebraic and Discrete Methods*, 5(3):287–293, 1984.

[49] D. Tompkins. UCBSAT – The stochastic local search SAT solver. `http://www.satlib.org/ubcsat/`.

[50] G.S. Tseitin. On the complexity of derivation in propositional calculus. In A.O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part II*, volume 8 of *Seminars in Mathematics, V.A. Steklov Mathematical Institute, Leningrad*, pages 115–125. Consultants Bureau, 1969. [Translated from Russian. Reprinted in J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967–1970*, pages 466–483. Springer, 1983.].

[51] A. Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, 1987.

[52] J.P. Warners and H. van Maaren. A two-phase algorithm for solving a class of hard satisfiability problems. *Operations Research Letters*, 23(3-5):81–88, 1998.

[53] N.C. Wormald. Models of random regular graphs. In J.D. Lamb and D.A. Preece, editors, *Surveys in Combinatorics, 1999*, number 267 in London Mathematical Society Lecture Note Series, pages 239–298. Cambridge University Press, 1999.

[54] zChaff 2004.5.13. `http://www.princeton.edu/~chaff/zchaff.html`.