

Operations Research based approaches for
the (maximum) satisfiability problem

Operations Research based approaches for the (maximum) satisfiability problem

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.dr.ir J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen

op maandag 13 februari 2006 om 13.00 uur

door Linda VAN NORDEN

doctorandus wiskunde
geboren te Rotterdam.

Dit proefschrift is goedgekeurd door de promotor:
Prof.dr.ir. H.J. Sips

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr.ir. H.J. Sips	Technische Universiteit Delft, promotor
Dr. H. van Maaren	Technische Universiteit Delft, toegevoegd promotor
Prof.dr.ir. J.F. Groote	Technische Universiteit Eindhoven
Prof.dr.ir. C. Roos	Technische Universiteit Delft
Prof.dr. E. Speckenmeyer	Universität zu Köln
Dr. M. Anjos	University of Waterloo
Dr. C. Witteveen	Technische Universiteit Delft
Prof.dr.ir. J.L.G. Dietz	Technische Universiteit Delft, reservelid

Preface

Chapter 3 of this thesis is based on a paper published in the Annals of Mathematics and Operations Research [142] and one published in the proceedings of the SAT2003 conference [141]. Chapter 4 is based on a paper published in the proceedings of the SAT2005 conference [144] and a paper submitted to Discrete Applied Mathematics [143].

My other publications include a paper on multi-product lot-sizing with a transportation capacity reservation contract published in the European Journal of Operational Research [146]. A paper on a winner determination problem of tendering transportation services will be published in the Zeitschrift für Betriebswirtschaft [148]. In the proceedings of the CP2002 conference a short abstract on the algorithm from Chapter 3 has appeared [147].

Contents

1	Introduction	1
1.1	The satisfiability problem	1
1.2	Central position of the satisfiability problem	3
1.3	Research contributions	7
1.4	Outline of the thesis	9
2	Related research	11
2.1	Satisfiability solvers	11
2.1.1	Simplification rules	11
2.1.2	The Davis-Putnam and the DLL method	12
2.1.3	Data structures	14
2.1.4	Complete satisfiability solvers	14
2.1.5	Incomplete satisfiability solvers	15
2.1.6	Preprocessing and symmetry	16
2.2	Horn clause related issues	17
2.2.1	Definitions of Horn clause related concepts	17
2.2.2	Variants and extensions	18
2.3	Phase transitions	19
2.3.1	Random k-SAT	19
2.3.2	Horn satisfiability	20
2.3.3	Hard problem instances	20
2.3.4	Phase transitions in some other problems	21
2.4	Maximum satisfiability	22
2.4.1	Approximation algorithms	22
2.4.2	Algorithms based on integer linear programming	23
2.4.3	Local search approaches	23
2.5	Graph-coloring	24
2.5.1	Satisfiability and graph coloring	24
2.5.2	Approximation algorithms	24
2.6	Linear Programming	25
2.7	Semidefinite Programming and Satisfiability	25
2.7.1	Semidefinite programming in general	25
2.7.2	Semidefinite programming methods for MAX- k -SAT	26

2.7.3	Proving unsatisfiability with semidefinite programming	27
2.7.4	Semidefinite programming algorithms	27
2.7.5	Sums of squares approximations	28
3	Correlations between reverse Horn fractions, satisfiability and solver performance	31
3.1	Introduction	31
3.2	Horn fractions	32
3.2.1	Weighted and MAX-Horn variants	33
3.2.2	Greedy Horn heuristic	34
3.3	Reverse Horn fraction and non-negative fraction	35
3.3.1	Optimal reverse Horn fraction and MAXSAT-fraction	36
3.3.2	Experiments on reverse Horn fraction and non-negative fraction	38
3.4	Correlation with computational effort	41
3.4.1	Estimation of average run time	41
3.4.2	Scaled run time distribution	42
3.4.3	Correlation between reverse Horn fraction and run time	43
3.5	Correlation between reverse Horn fraction and	45
3.5.1	SurveyPropagation	45
3.5.2	UnitWalk	46
3.6	Correlation between reverse Horn fraction and satisfiability	46
3.7	Graph-3-coloring	49
3.8	The algorithm aHS	52
3.8.1	Failed literal tests	53
3.8.2	Linear Program	53
3.8.3	Search tree design	53
3.8.4	The objective of the linear program	55
3.9	Conclusions	57
4	Sums of Squares based approximation algorithms for MAX-SAT	59
4.1	Introduction	59
4.2	SDP-based upper bounds for MAX-2-SAT	66
4.2.1	Comparison of SOS_{GW} and Goemans-Williamson approach	67
4.2.2	Adding valid inequalities vs adding monomials	69
4.2.3	SOS_{ap} on worst known case for Feige-Goemans	72
4.2.4	Experimental results for random MAX-2-SAT	73
4.3	Complexity of different approaches wrt	75
4.4	SDP-based upper bounds for MAX-3-SAT	78
4.4.1	Karloff-Zwick inequalities vs monomials in SOS_{pt}	79
4.4.2	Experimental results for random MAX-3-SAT	80
4.5	Experimental results for SOS_t and SOS_{pt} on 3-SAT	80
4.6	Rounding procedures based on SOS_p and SOS_t	82
4.6.1	Skewed rounding on MAX-2-SAT instances based on SOS_p	84

4.6.2	A rounding procedure for MAX-3-SAT based on SOS_t	87
4.7	Fraction of unit constraints in SOS_p , SOS_t and SOS_{pt}	90
4.8	Counterexamples to uncovered cases of Hilbert's	91
4.9	Conclusions	92
5	Forbidden color problem instances	93
5.1	Transforming graph coloring to satisfiability	94
5.2	Preliminary definitions	94
5.3	A generator for forbidden color instances	95
5.4	Experiments on forbidden color instances	96
5.4.1	Experiments on small instances	96
5.4.2	Scaling to larger Graph-3-SAT instances	98
5.5	Structured graph coloring instances	100
5.6	Conclusions	102
6	Conclusion	103
	Bibliography	104
	Summary	119
	Samenvatting	121

Chapter 1

Introduction

The satisfiability problem is an important problem in theoretical computer science because of its central position in complexity theory. The satisfiability problem was the first problem proven to be \mathcal{NP} -complete by Cook [29]. The concept \mathcal{NP} -complete is explained in the short introduction in complexity theory in Section 1.2.

Each instance of an \mathcal{NP} -complete problem can be translated into an instance of the satisfiability problem in polynomial time. This makes it very important to develop fast solvers and good approximation algorithms for satisfiability.

Satisfiability formulations are used in many different fields as is described in Section 1.2. Due to the good performance and efficiency of currently available satisfiability solvers, satisfiability solving procedures are becoming interesting alternatives for an increasing number of problem domains. An increasing number of researchers is interested in satisfiability-based research, and there are nowadays annual conferences specially devoted to satisfiability.

This chapter gives the basic definitions illustrated by an example. The central position of the satisfiability problem is discussed and the research contributions of this thesis are mentioned.

1.1 The satisfiability problem

In this subsection, we give a description of the satisfiability problem, give some definitions and an example. Let x_1, \dots, x_n be a set of *propositional variables* that can be assigned the truth values ‘true’ and ‘false’. Logic formulas are constructed with propositional variables and the operators \wedge (and), \vee (or) and \neg (not). $x_1 \vee x_2$ means that either x_1 or x_2 or both needs to be true. $x_1 \wedge x_2$ requires both x_1 and x_2 to be true. $\neg x_1$ means that x_1 has to be false. A *literal* is an atomic propositional variable or its negation. The disjunction of the literals x_1, \dots, x_n is

$$x_1 \vee x_2 \vee \dots \vee x_n.$$

A *clause* is a disjunction of literals. The *length* of a clause is the number of different literals in that clause. A clause of length 1 is called a *unit clause*. A clause is *Horn* if

at most one literal is positive. A conjunction of clauses C_1, C_2, \dots, C_m is

$$C_1 \wedge C_2 \wedge \dots \wedge C_m.$$

A *CNF-formula* is a conjunction of clauses. CNF is an abbreviation of Conjunctive Normal Form. The number of variables in a CNF-formula is denoted by n , the number of clauses by m .

Definition 1.1.1. *The satisfiability problem is to assign truth values true and false to the propositional variables such that all clauses in the CNF-formula are true or to prove that such a truth assignment does not exist.*

Example

In order to clarify these definitions, we will start with an example on how to formulate a satisfiability problem. In this example we have a group of musicians that play different instruments: oboe, bassoon, recorder, and harpsichord. The goal is to divide these musicians into a number of smaller groups to play chamber music. The division of the musicians over the groups has to satisfy a number of constraints on the compositions of the groups. Let \mathcal{M} be the set of musicians, containing a number of oboists h_1, \dots, h_H , bassoon players f_1, \dots, f_F , recorder players b_1, \dots, b_B and harpsichord players c_1, \dots, c_C . Let E be the number of chamber music ensembles to compose and \mathcal{E} the set of ensembles. We introduce variables x_{ij} that indicate whether player i is a member of ensemble j or not.

We give some examples of possible constraints.

- Each musician i is a member of at least one ensemble.

$$x_{i1} \vee x_{i2} \vee \dots \vee x_{iE}$$

- Musician i cannot be in two ensembles. Note that we can formulate this by requiring for each pair of ensembles e_1 and e_2 that the musician is at least not in one of the two.

$$\neg x_{i,e_1} \vee \neg x_{i,e_2}, e_1, e_2 \in \mathcal{E}, e_1 \neq e_2$$

- In each ensemble j there is at least one person playing harpsichord.

$$x_{c1,j} \vee x_{c2,j} \vee \dots \vee x_{cC,j}.$$

- If there is an oboist h_i in ensemble j , there should be at least one person playing bassoon in ensemble j .

$$\neg x_{h_i,j} \vee x_{f1,j} \vee \dots \vee x_{fF,j}$$

In this way, we can add many more constraints on the composition of the ensembles. The satisfiability problem is in this case the question whether there exists a division of the musicians over the groups that satisfies each of the constraints. Or, mathematically, whether there exists a truth assignment to the variables $x_{ij}, i \in \mathcal{M}, j \in \mathcal{E}$ satisfying all clauses. \square

A formula is *satisfiable* if a satisfying truth assignment exists and *unsatisfiable* otherwise. *k-SAT* is defined as the class of problems where the CNF-formula consists only of clauses with at most k literals. We denote the class of problems where the CNF-formula has only clauses with exactly k literals with $\{k\}$ -SAT. Cook [29] proved that the class of k -SAT problems, $k \geq 3$, is \mathcal{NP} -complete. However, 2-SAT is solvable in polynomial time. Del Val [43] presents a linear time algorithm for 2-SAT based on unit resolution, which is defined in Section 2.1. *MAX-SAT* is the problem of finding the maximal number of clauses that can be satisfied in a CNF-formula.

The density of a CNF-formula plays an important role in the research on phase transitions for the satisfiability problem. Also in Chapter 3 the density is a frequently used concept. The density of a k -SAT formula is the number of clauses divided by the number of variables.

Techniques from Operations Research are essential to the algorithms proposed in Chapter 3 and Chapter 4. In Chapter 3 a linear program based on the CNF-formula is used for designing the search tree of the algorithm aHS that is used for computing the newly introduced parameter k -level reverse Horn fraction. We believe this linear program is an essential element to create a search tree yielding fast converging k -level reverse Horn fractions. For the main algorithm from Chapter 4, computing upper bounds on the MAX-SAT solution, semidefinite programming is its main ingredient.

1.2 Central position of the satisfiability problem

This subsection describes a number of applications in which satisfiability (based) formulations are used. Besides, a short introduction on complexity is given. Cadoli and Schaerf [23] present a method to automatically translate problem specifications from \mathcal{NP} -complete problems into CNF-formulas. Their method uses as input a formulation of the problem into the logic-based language NP-SPEC and automatically constructs a CNF-formula. They tested their method on the graph coloring, Hamiltonian cycle, and job-shop scheduling problems. Besides for solving problems, the method can also be used for generating benchmark instances for satisfiability solvers.

Planning and scheduling

Satisfiability methods for solving planning and scheduling problems have attracted quite some attention. It started with the paper by Kautz and Selman [88]. Ernst, Millstein and Weld [52] propose a method to automatically generate satisfiability encodings for planning problems formulated in the generally used STRIPS language. Some other examples of using a satisfiability-based approach for planning and scheduling include

[24], [47], [87], [89], [121], [123], [151]. Crawford and Baker [33] apply satisfiability algorithms to scheduling problems.

SATPLAN-04 [86] is a satisfiability-based planner using ideas from [88]. The fact that SATPLAN-04 won the first prize for optimal deterministic planning in the International Planning Competition of the International Conference on Automated Planning and Scheduling in 2004 illustrates that in certain areas satisfiability-based approaches are competitive with other Artificial Intelligence and Operations Research approaches. Also Rintanen [122] describes a class of planning instances that can be solved more efficiently with a planner based on satisfiability algorithms than with two planners using a heuristic local search procedure.

Example To give some idea about how planning problems can be formulated as satisfiability instances, we will give an example. In this example, we have L locations. An order is the task to bring a package from its load location to its destination. There is a truck with unlimited capacity that can perform the orders. It takes one unit of time to go from any location to any other location. The goal is to find a route such as to minimize travel time such that all packages are transported from their load location to their destination. The variables introduced are

- TM_{xyt} being 1 if the truck goes from location x to y at time t , 0 otherwise
- TA_{xt} being 1 if the truck is at location x at time t , 0 otherwise
- PM_{pxyt} being 1 if package p goes from location x to y at time t , 0 otherwise
- PA_{pxt} being 1 if package p is at location x at time t , 0 otherwise

Below, we give examples of different types of clauses in a possible satisfiability formulation of this problem.

- If the truck moves from x to y at t , it has to be at x at t . This type of clauses says that an action implies its preconditions.

$$\neg TM_{x,y,t} \vee TA_{x,t}$$

- If the truck moves from x to y at t , it is at y at time $t + 1$. This type of clauses says that an action implies its effects.

$$\neg TM_{x,y,t} \vee TA_{y,t+1}$$

- If package p moves from x to y at t , the truck has to move from x to y at time t , because the package can only move inside the truck.

$$\neg PM_{x,y,t} \vee TM_{x,y,t}$$

- The truck cannot be at two locations at the same time.

$$\neg TA_{x,t} \vee \neg TA_{y,t}, x \neq y$$

- The truck can only be at location x at $t + 1$ if it was already there at t or has moved to it at time t .

$$\neg TA_{x,t+1} \vee TA_{x,t} \vee \bigvee_{y \neq x} TM_{y,x,t}$$

In this way, many more constraints should be added to this example to completely formulate the problem, but we think the above constraints show sufficient insight in how satisfiability formulations can be constructed for planning problems. \square

Operations research

Techniques from Operations Research have contributed to the satisfiability research observably as can be seen in Chapter 2 and the references therein and of course, in Chapter 3 (linear programming) and Chapter 4 (semidefinite programming).

Satisfiability-based approaches are also used to solve Operations Research problems. Currently, even widely used introductory textbooks on Operations Research like the 8-th edition of the book of Hillier and Liebermann [73] devote some attention on the combination of ideas from satisfiability-related research and Integer Programming. As is clear from the book of Hooker [75] the ideas from research on logic and satisfiability can be used to design optimization algorithms. The research field of Constraint Programming uses ideas from satisfiability and logic.

Warners [150] presented a method to transform a 0-1 linear programming problem with integral coefficients into a satisfiability problem and showed that in some cases this satisfiability problem can be solved in a period of time comparable to well-known operations research techniques on the 0-1 linear program itself. Also in more indirect ways, the results in satisfiability research might contribute to operations research. For example, Manquinho, Marques Silva, Oliveira and Sakallah [101] use insights from research on satisfiability solvers for designing a solver for 0-1 integer linear programs.

Complexity theory

A very short introduction on complexity theory explains what the term \mathcal{NP} -complete means. The class of \mathcal{NP} -complete problems is a subclass of the class \mathcal{NP} . The letters \mathcal{NP} stand for 'non-deterministic polynomial time'. The class \mathcal{NP} contains decision problems that can be answered by 'yes' or 'no'. For each problem in \mathcal{NP} , there exists for a 'yes'-answer a so-called certificate that can be used to check the correctness of the yes-answer in polynomial time. It is trivial to show that a yes-answer to an instance of the satisfiability problem is correct by giving a truth assignment satisfying all clauses as a certificate. For problems in the class \mathcal{NP} it is only necessary that a certificate can be checked in polynomial time, not that one can find a certificate in polynomial time.

The \mathcal{NP} -complete problems are by definition the hardest problems in the class \mathcal{NP} . A problem Q is \mathcal{NP} -complete if each problem in \mathcal{NP} can be reduced to Q in

polynomial time. A polynomial time reduction from Q' to Q is a polynomial time algorithm R that translates any instance of Q' into a corresponding instance of Q . As a consequence, if any of the \mathcal{NP} -complete problems can be solved in polynomial time, all problems in \mathcal{NP} can be solved in polynomial time. Formulated in a different way, if any of the \mathcal{NP} -complete problems can be solved in polynomial time, the class \mathcal{NP} would be equal to the class \mathcal{P} of problems that can be solved in polynomial time. It is generally assumed that this is not the case. This is the famous assumption $\mathcal{NP} \neq \mathcal{P}$.

Let Q_p be problem Q restricted to instances such that the largest integer k in the instance is smaller than or equal to $p(SE)$, with SE the length of the encoding of the instance. A problem Q is strongly \mathcal{NP} -complete if Q_p is \mathcal{NP} -complete for some polynomial $p(x)$.

Digital systems

The second field in which satisfiability-based approaches are frequently studied is digital systems. Marques-Silva and Sakallah [103] propose a satisfiability-based algorithm for Automatic Test Pattern Generation (ATPG). ATPG algorithms are tools for generating test inputs for fault detection in digital circuits after they have been produced. A digital circuit containing connected OR, AND, NOT, NAND and NOR gates can be represented by a CNF-formula. The variables represent the inputs and outputs of the gates. For example $X = \text{NOT}(Y)$ can be represented by the clauses

$$(X \vee Y) \wedge (\neg X \vee \neg Y)$$

. In this way all type of gates can be formulated as sets of clauses. The formula for detecting a particular fault contains a.o. clauses describing the good circuit, clauses describing the faulty circuit and variables expressing the difference.

Other applications of satisfiability in electrical engineering problems include the algorithms for circuit delay computation by Guerra e Silva, Marques-Silva and Silveira [64] and Chen and Gupta [27] and the one by Veneris [132] for design diagnosis (i.e. finding errors in a faulty chip).

Register allocation

In computer science, a satisfiability-based approach is used for register allocation by Potlapally [118]. Register allocation is the problem to assign variables in a computer program to registers in the processor such that the execution time of the program is minimal. There are generally more variables than registers, and the variables have to be assigned to the registers such that as few as possible variables sharing a register are active at the same time. Sharing a register by two active variables requires moving variables to a temporary location, which costs time.

Combinatorics

A completely different application of satisfiability is presented by Dransfield, Liu, Marek, and Truszczynski [49]. They show that the problem of computing Van der Waerden numbers can be represented as a satisfiability problem. Van der Waerden numbers are positive integers W such that every partition of the numbers $\{1, \dots, W\}$ into k parts has at least one part with an arithmetic progression of length l . An arithmetic progression is a sequence of l numbers $c + id$, $i = 0, \dots, l - 1$ such that the differences between successive terms is a constant d . Herwig, Heule, van Lambalgen and van Maaren [71] present a satisfiability based method to compute lower bounds for the Van der Waerden number which are better than the best known ones till that moment.

Cryptography

Massacci and Marraro [104] use satisfiability solvers for testing the properties of cryptographic algorithms and keys. They present a method that can be used to model and verify state-of-the-art cryptographic algorithms like the widely used Data Encryption Standard. Finding a satisfying assignment for the CNF-formulae concerned is equivalent to recovering a key in a cryptanalytic attack. At the time of their research, the instances coming from state-of-the-art cryptographic algorithms were too hard for the satisfiability solvers tested. Fiorini, Martinelli and Massacci [54] study satisfiability formulations for public-key cryptography. The problem they model as a CNF-formula is: given three numbers a , b and c , find a number f such that $c \equiv f^a \pmod{b}$. The CNF-formulas coming from cryptographic problems provide hard, random, and structured benchmarks for satisfiability solvers.

Other applications

Even in research fields as far away from computer science and mathematics as medicine, satisfiability-based approaches can be useful. Di Giacomo, Felici, Maceratini and Truemper [46] use a variant of the satisfiability problem minimizing the costs assigned to variables being true for diagnosing carcinoma. Based on a number of features the method is able to separate patients with and without hepatocellular carcinoma. It can be used to detect the disease in an early stage. Satisfiability solvers can even be used for solving the popular Sudoku-puzzles as illustrated on [133].

1.3 Research contributions

This section describes the main research contributions of this thesis.

- This thesis introduces an enhanced version of the Horn fraction, called k -level reverse Horn fraction and an algorithm for computing it. The experiments show

that for fixed density random $\{3\}$ -SAT instances a correlation between the k -level reverse Horn fraction and the computational effort of a number of complete and incomplete satisfiability solvers exists. Based on the k -level reverse Horn fraction of an instance, an estimation could be made of the time it takes to solve it in case it would be satisfiable and unsatisfiable.

- Extensive experiments with fixed-density random $\{3\}$ -SAT CNF-formulas show that there exists also a correlation between the k -level reverse Horn fraction and the fact whether an instance is satisfiable or not. This correlation is not only present for the smaller instances but remains observable also for large instances. This correlation is even more prominent for CNF-formulas coming from graph-3-coloring instances. The correlation between this k -level reverse Horn fraction and the satisfiability of the instances is important for two reasons. First, this correlation might be used to estimate the probability an instance is satisfiable based on its k -level reverse Horn fraction. Secondly, for random $\{3\}$ -SAT instances at the density threshold, for which about 50% of the instances is satisfiable, we have a phase transition with respect to a second parameter, the k -level reverse Horn fraction.
- A sums of squares (SOS) method based on semidefinite programming is presented which gives an upper bound on the MAX-SAT solution. The semidefinite program makes use of a basis of monomials to decompose the polynomial. We study the results of the method with different monomial bases. Furthermore, a rounding procedure is developed that computes lower bounds on the MAX-SAT solution based on the proposed semidefinite program.
- With the smallest of the monomial bases, M_{GW} , the upper bound of the SOS approach is proven to be equal to the upper bound obtained by the famous method of Goemans and Williamson [60] for MAX-2-SAT. The SOS approach with the larger monomial basis M_p is shown to give for MAX-2-SAT at least as good upper bounds as the method of Feige and Goemans [53]. The theoretical results are supported by experimental evidence on a set of MAX-2-SAT instances. The observed performance guarantee of a method is defined as the average number of clauses satisfied over the runs of the rounding procedure divided by the upper bound obtained by the method considered. The SOS approach combined with the rounding procedure gave a better observed performance guarantee for MAX-2-SAT than the method of Goemans and Williamson.
- Experimental results have illustrated that the SOS-approach with the monomial basis M_{pt} gives for MAX-3-SAT a better observed performance guarantee than the algorithm of Karloff and Zwick. The SOS approach with monomial basis M_{pt} finds a proof of unsatisfiability for more instances than the rank-3 relaxation of Anjos [4] for a set of 3-SAT instances with 20 variables.
- A method has been presented to generate so-called forbidden color instances that

are experimentally shown to be more difficult than the graph coloring instances they originate from.

1.4 Outline of the thesis

Chapter 3 studies the correlation between an enhanced concept of the Horn fraction, satisfiability, and the performance of both complete and incomplete satisfiability solvers. We justify experimentally that this enhanced Horn fraction is correlated more strongly with satisfiability and solver performance than a sub-optimal approximation of the MAX-SAT fraction, i.e. the fraction of clauses satisfied in an optimal MAX-SAT solution. The correlation with the performance of a number of state-of-the-art complete solvers on random $\{3\}$ -SAT is studied in Section 3.4, and Section 3.5 discusses the correlation with the performance of two incomplete solvers. Section 3.6 presents experiments on the correlation for random $\{3\}$ -SAT between the enhanced Horn fraction and satisfiability. These correlations do not only exist for random $\{3\}$ -SAT; from the experiments presented in Section 3.7 it can be concluded that these correlations are even more prominent in random graph-3-coloring instances. The algorithm aHS developed for computing the enhanced Horn fraction is described in Section 3.8.

Chapter 4 presents a new method for computing upper and lower bounds for MAX-SAT based on a sums-of-squares approach. The minimum of a polynomial F can be approximated by the largest value ϵ such that $F - \epsilon$ is a sum of squares. This ϵ can be found in polynomial time (up to a prescribed precision) by solving a semidefinite program. In this way, it is possible to obtain an upper bound on the MAX-SAT solution by using a polynomial that expresses the number of clauses that is not satisfied by an assignment. In Section 4.2, we prove by means of a primal-dual argument that our SOS (Sums of Squares)-based approach with only monomials of degree one gives upper bounds for the MAX-2-SAT problem that are the same as the upper bounds of the famous algorithm of Goemans and Williamson [60]. Adding monomials of degree two to the monomial basis of the SOS-based semidefinite program yields upper bounds at least as tight as those from adding corresponding valid inequalities of Feige and Goemans [53] to the semidefinite program (SDP) of Goemans and Williamson. In section 4.3, the complexities of the different methods are evaluated. In Section 4.4, the upper bounds of the SOS approach with different monomial bases are compared for MAX-3-SAT.

Experimental results in Section 4.5 illustrate that unsatisfiability can be proved observably more often by upper bounds obtained by the SOS-approach with the relevant monomials of degree one, two and three than for the SDP of Anjos [5] for a randomly selected set of 3-SAT instances with 20 variables and densities from 4.0 to 4.3. For the selected instances with up to 15 variables, both SDPs performed equally well. In Section 4.6 a new rounding procedure for obtaining lower bounds for MAX-SAT is presented. The observed performance guarantee of a method is defined as the average number of clauses satisfied over the runs of the rounding procedure divided by the upper bound obtained by the method considered. Our experiments illustrate that the observed per-

formance guarantee of the SOS-based method with the relevant monomials of degree two is better than the observed performance guarantee of the algorithm by Goemans and Williamson and the proven performance guarantee of the method of Lewin, Livnat and Zwick [95] for MAX-2-SAT. In the experiments on MAX-3-SAT instances with 20 variables, we obtained a better observed performance guarantee with the SOS-based approach with the appropriate monomials of degree one, two and three than with the algorithm of Karloff and Zwick [84]. In Section 4.7, an experimental evaluation of the structure of the SOS SDPs is presented.

Chapter 5 studies a method for generating a class of instances that are more difficult to solve for state-of-the-art satisfiability solvers than random graph- k -coloring instances converted into satisfiability. These so-called forbidden color instances are made by adding unit clauses to the satisfiability formulation of graph- k -coloring problems.

Chapter 2

Related research

This chapter gives an overview of the literature on topics related to this thesis.

2.1 Satisfiability solvers

In recent years, many algorithms have been developed and implemented to solve the satisfiability problem. In this section, we discuss some well-known simplification rules, the DLL method and the importance of efficient data structures for satisfiability solvers. Furthermore, we give a short overview of a number of (relatively) recent complete and incomplete satisfiability solvers.

A satisfiability solver is *sound* if every instance for which it returns 'yes' is indeed satisfiable. A solver is *complete* if it returns 'yes' for every satisfiable instance. In other words, if it does not return 'yes', one can be sure the instance concerned is unsatisfiable. In recent years, a number of competitions for satisfiability solvers has been organized. The goal is to enable a good comparison between the performance of different satisfiability solvers on different types of problem instances. Over last years, a huge improvement of the performance of the best solvers has been observed.

2.1.1 Simplification rules

Most satisfiability solvers use some kind of unit propagation. *Unit propagation* is the repeated application of unit resolution and unit subsumption of variables occurring in unit clauses until no unit clauses remain. *Resolution* is a simplification rule for logical formulas. A simple example illustrates how it works. If resolution gets as input the clauses $Y \vee X_1 \vee X_2$ and $\neg Y \vee X_3 \vee X_4$, it outputs $X_1 \vee X_2 \vee X_3 \vee X_4$. Resolution gives the result of the conjunction of the two clauses. Unit resolution is a special case of resolution. The unit clause X is combined with every clause containing $\neg X$, which results in a CNF-formula with just all occurrences of the literal $\neg X$ removed. Unit subsumption of a literal X in a CNF-formula removes all clauses containing the literal X , which are satisfied if X is true. Zhang and Stickel [154] present an efficient algorithm for unit propagation.

The *pure literal rule* implies that if a variable occurs only positively or negatively, it can be fixed to true or false, respectively. For example, if the literal X occurs in a CNF-formula but the literal $\neg X$ does not occur in the formula, the variable X can be fixed to true.

Equivalent literal detection identifies a set of literals that must have the same truth value or yield a contradiction. For example, consider a CNF-formula that contains among its clauses the two clauses

$$\begin{aligned}x_1 \vee \neg x_2, \\ \neg x_1 \vee x_2.\end{aligned}$$

If x_1 is true, x_2 has to be true to satisfy these two clauses. On the other hand, if x_1 is false, then x_2 has to be false too. This implies that x_1 and x_2 must have the same truth assignment in any satisfying assignment. In that case, it is possible to replace all occurrences of x_2 by x_1 and remove these two clauses to reduce the size of the CNF-formula.

2.1.2 The Davis-Putnam and the DLL method

The Davis-Putnam method

Davis and Putnam [38] present a method for solving satisfiability based on complete resolution and using the pure literal rule and unit propagation. Complete resolution is the process of applying resolution as long as there exist variables that occur both positive and negative. In each step, one variable X is selected and each clause with the literal X is combined with each clause containing the literal $\neg X$. The number of clauses generated in each step equals the number of clauses containing X times the number of clauses containing $\neg X$. If this process stops, each variable left occurs only positive or only negative. If the instance is unsatisfiable, an empty clause appears. Otherwise, the variables can be set according to whether they occur positive or negative. This method is not very efficient because clauses are generated independently on whether they are necessary to find a satisfying assignment or prove unsatisfiability. The number of generated clauses might be exponential.

The DLL method

A number of current complete solvers, like OKsolver, chaff and knfs, are based on the Davis-Logemann-Loveland (DLL) method [37]. The DLL method is based on unit propagation and binary branching. Branching is done on a variable being true or false. The three most important choices when designing a DLL-based algorithm are the choice of the method for determining which variable to branch on, how to backtrack, and which rules are to be applied to deduce necessary assignments.

We will now shortly discuss each of these aspects.

- **Deducing necessary assignments.** The original DLL-algorithm only uses unit propagation and the pure literal rule to detect necessary assignments, but more rules might be used.
- **Backtracking decisions.** Backtracking can be done to the parent of the node where a conflict is found, but there exist more efficient backtracking rules that backtrack to the source of the conflict, skipping assignments that are irrelevant for the conflict.
- **Branching decisions.** Some of the DLL-based satisfiability solvers use a look-ahead procedure to determine which variable is to be used next for branching based on some evaluation criterion, for example the number of clauses of length two. For each variable X the look-ahead procedure evaluates the criterion for $F \vee X$ and $F \vee \neg X$ with F the CNF-formula. Using the look-ahead procedure, one might discover so-called failed literals, i.e. if $F \vee X$ or $F \vee \neg X$ is unsatisfiable, X can be fixed to false or true respectively. Li and Anbulagan [96] present a simple unit propagation based heuristic for selecting branching variables that performs well when used in a DLL procedure.

Now, we will give a short overview of the structure of a DLL-based algorithm.

Algorithm 1: DLL-based algorithm

Input: A CNF formula.

Output: Satisfiable or unsatisfiable

- (1) Select a variable that has not been assigned a truth value.
- (2) **if** this is not possible and all clauses are satisfied,
- (3) go to step 14.
- (4) **else**
- (5) Deduce all implied assignments that can be obtained with the selected simplification rules.
- (6) **if** no conflict is found
- (7) go back to step 1, i.e. one level deeper in the search tree.
- (8) **else**
- (9) Search for the cause of the conflict and identify to which level d must be backtracked to resolve the conflict.
- (10) **if** $d = 0$,
- (11) **return** Unsatisfiable
- (12) **else**
- (13) backtrack to level d and undo all changes made in levels $d' > d$.
- (14) **return** Satisfiable.

An implementation of a DLL-based algorithm is described by Zhang and Stickel [155]. A more extensive introduction on DLL-based algorithms for satisfiability can be found in the PhD-thesis of Lynce [98].

2.1.3 Data structures

For implementing a fast satisfiability solver it is very important to use highly efficient data structures. The authors of Chaff [109] propose a very efficient data structure for use in a DLL-based satisfiability solver: the watched-literal structure. The main idea of this data structure is that of each clause at most two literals are being watched, i.e. only these two literals are considered when updating the formula after an assignment. If one of the two watched literals is assigned and the clause is not satisfied, then if in existence, a new unassigned literal is selected to be watched. Unit clauses are detected if no second free literal is left to be watched. Van Gelder [139] proposes and analyzes a generalization of this watched literal structure that also supports equivalent-literal detection and two-literal clause detection. Lynce and Marques-Silva [99] analyze a number of existing data structures for backtrack search satisfiability solvers and propose new efficient data structures.

2.1.4 Complete satisfiability solvers

The most important aspect of a complete satisfiability solver is that it returns 'yes' for every satisfiable instance and provides a certificate of unsatisfiability for unsatisfiable instances.

- The **OKsolver** [93] is a DLL-based algorithm, with reduction by failed literals and autarkies. An autarky is a set A of literals such that every clause that contains the negation of a literal in A also contains a literal from A . The branching heuristic chooses a variable, creating as many new two-literal clauses as possible, and the first branch is chosen maximizing an approximation of the probability that the subproblem is satisfiable. The OKsolver won the award for the best solver in the random category in the SAT-competition in 2002. It still belongs to the best solvers in this category but also performs quite well on different types of instances.
- **knfs** [45] is a complete DLL-based satisfiability solver with a very effective branching heuristic based on the search for variables in the backbone of the CNF-formula. A variable belongs to the backbone of a CNF-formula if it has the same value in every solution that satisfies the maximum number of clauses. knfs won the award for the best solver on unsatisfiable instances in the random category in the SAT-competition in 2003. It is designed and optimized to perform well in this category.
- **Chaff** [109] introduces the very efficient watched literal structure for determining when a clause becomes a unit clause. A second important aspect of chaff is learning. Learning is the storage of information obtained in a certain part of the search space that might also be used in other parts of the search space. Learning adds clauses to the existing set of clauses to reduce the search space. Chaff was

the winner of the SAT competition in 2004 in the industrial category on both unsatisfiable and a combination of satisfiable and unsatisfiable instances.

- The solver **march_eq** [72] is a look-ahead solver. One of the most important features of this solver is a cheap integration of equivalence reasoning and local learning in the look-ahead procedure. Other incorporated procedures like adding constraints resolvents are only particularly useful for specific problem instances. March_eq won in the SAT-competition in 2004 in the crafted category on unsatisfiable and the combination of satisfiable and unsatisfiable instances.
- **Berkmin** [62] incorporates some new properties besides well-working aspects inherited from previous solvers like for example restarts. The idea behind restarts is that one might make an unhappy choice for one of the branching variables early on in the decision tree which might lead to long computation times. Running the algorithm up to a certain cut-off, and then restarting with different branching decisions might lead to much quicker solutions for satisfiable instances. One of the new features is a new clause database management system to store conflict clauses obtained by learning. In addition they use a different heuristic to decide which of the two branches is explored first at a node where branching has to take place. The removal of clauses from the database of learned clauses is not only based on length but also on 'age' and 'activity'.

2.1.5 Incomplete satisfiability solvers

Incomplete satisfiability solvers are very fast in solving large satisfiable instances but do not have the ability to prove unsatisfiability. Many incomplete satisfiability solvers are stochastic local search algorithms. Almost every local search algorithm for satisfiability and MAX-SAT starts with an initial assignment and moves randomly to a new assignment by flipping one or more variables. UBCSAT [137] is an experimentation environment providing efficient implementations of a number of well-known stochastic local search algorithms for satisfiability and MAX-SAT, like for example GSAT [129], WalkSAT [128] and AdaptNovelty+ [76].

- **UnitWalk** [74] is an incomplete satisfiability solver combining local search and unit clause elimination. The algorithm consists of periods in each of which an assignment is built. In each period, a random permutation of the variables is obtained. If there are any unit clauses, one of them is selected and the variable in it is fixed to the corresponding truth value. If no unit clauses exist, the first unassigned variable from the permutation is chosen to be fixed. In the SAT-competition in 2003, UnitWalk won the award in the satisfiable random benchmarks category.
- **WalkSAT** [128] proceeds from one assignment to the next by randomly selecting a clause that is not satisfied and then choosing randomly or greedily a variable from the clause to be flipped.

- **Adaptnovelty+** [76] is a very effective incomplete satisfiability solver. The performance of stochastic local search algorithms like WalkSAT [128] depends heavily on the value of the noise parameter. This parameter needs to be fine-tuned manually and the best performance is obtained for different values for different classes of problem instances. Adaptnovelty+ contains a self-tuning noise mechanism, which avoids the need to set this parameter manually. Adaptnovelty+ was the winner in the random category in the SAT-competition in 2004.
- **SP(SurveyPropagation)** [22] is an incomplete solver completely different from the other ones discussed in this subsection. It is based on ideas from statistical physics and especially designed for solving very large instances. It uses an improved message-passing algorithm sending messages about the probability of being satisfiable between variables and clauses to solve instances close to the threshold of 3-SAT, which have clusters of solutions that are not very similar and many clusters of partial solutions. The authors report their algorithm to be able to solve instances with 10^6 variables.

2.1.6 Preprocessing and symmetry

Preprocessing a CNF-formula before feeding it into a satisfiability solver might considerably reduce computation time. One of the characteristics of CNF-formulas that often leads to rather long computation times is symmetry in the variables. A symmetry is a permutation of the variables such that each clause maps into a clause with the same number of positive and negative literals. For example, the permutation $(x_1y_1)(x_2y_2)$ is a symmetry for the CNF-formula consisting of the clauses

$$\begin{aligned} &x_1 \vee x_2, \\ &y_1 \vee y_2, \\ &\neg x_1 \vee \neg y_1, \\ &\neg x_2 \vee \neg y_2. \end{aligned}$$

CNF-formulas coming from graph-coloring instances heavily suffer from symmetry by construction. If a coloring exists, one can permute the colors in every possible way to obtain a feasible coloring with the same number of colors. Unfortunately this also holds for partial colorings not leading to a full coloring with the required number of colors. Hence, general-purpose satisfiability solvers may spend lots of time in searching parts of the search space that are equivalent except for a permutation of the colors. Another example of symmetry in CNF-formulas includes logistic planning problems involving a number of equivalent trucks.

Aloul, Markov and Sakallah [2] developed a procedure called Shatter that detects symmetries in a CNF-formula and adds symmetry-breaking clauses to the CNF-formula. The resulting formula is expected to be solved more efficiently by a general-purpose satisfiability solver. Sabharwal [125] designed the satisfiability solver SymChaff based on Chaff that uses dynamically maintained information about symme-

tries to speed up the search. His experiments illustrate that on certain problems with symmetries SymChaff is exponentially faster than Chaff.

Li, Jurkowiak and Purdom [97] describe how a symmetry-breaking schema can be incorporated in a DLL-based algorithm. Their experiments illustrate that even simple symmetry-breaking approaches might considerably reduce computation time.

2.2 Horn clause related issues

In this section, we give definitions of some Horn clause related concepts and mention some results from literature related to Horn clauses. Horn clause related concepts play a central role in Chapter 3. Horn clauses have besides nice theoretical properties also practical relevance. Selman and Kautz [127] propose a method to approximate CNF-formulas by Horn formulas. The set of satisfying assignments of the original CNF-formula is bounded from below and from above by two Horn formulas. Because Horn formulas can be solved in linear time, this gives a fast approximation to the original CNF-formula.

2.2.1 Definitions of Horn clause related concepts

In this subsection, we give definitions of four Horn clause related concepts and references to some of the key results on this topic.

Definition 2.2.1. *A clause is Horn if at most one literal is positive. A clause is reverse Horn if at most one literal is negative.*

Definition 2.2.2. *The (reverse) Horn fraction $H(\mathcal{F})$ of a CNF-formula \mathcal{F} is the number of (reverse) Horn clauses divided by the total number of clauses.*

Renaming, or flipping, a variable X means that every occurrence of X is replaced by $\neg X$ and every $\neg X$ is replaced by X . Renaming can change the (reverse) Horn fraction of a CNF-formula. Consider for example, the CNF-formula consisting of the clauses

$$\begin{aligned} X \vee Y \\ \neg X \vee \neg Z \\ X \vee \neg Y \vee Z \end{aligned}$$

The second clause is Horn, and the first and third clauses are not Horn, giving a Horn fraction of $\frac{1}{3}$. After renaming variable X , each of the clauses is Horn, resulting in a Horn fraction of 1. When renaming is involved, Horn and reverse Horn are equivalent concepts: by renaming all variables, all Horn clauses get reverse Horn clauses and vice versa.

Recognizing whether a formula \mathcal{F} is renamable Horn, i.e. whether all clauses can be made Horn by renaming, can be done in linear time, for example by the algorithm of

Aspvall, Plass and Tarjan [8]. *Horn satisfiability* is the restriction of the satisfiability problem to instances that have only Horn clauses. k -Horn satisfiability is the Horn satisfiability problem restricted to formulas with only clauses of length at most k . CNF-formulas with only Horn clauses can be solved in time linear in the number of literals by the algorithm of Dowling and Gallier [48] or the one of Del Val [43].

Definition 2.2.3. *The optimal (reverse) Horn fraction of a CNF-formula \mathcal{F} after renaming (in short optimal (reverse) Horn fraction), $H_{\text{opt}}(\mathcal{F})$, is the largest fraction of (reverse) Horn clauses that can be obtained by renaming variables in \mathcal{F} .*

The problem of computing the optimal Horn fraction after renaming is \mathcal{NP} -hard as is proven by Crama, Ekin and Hammer [32].

Definition 2.2.4. *Given a CNF-formula \mathcal{F} and a deterministic renaming routine \mathcal{R} , the sub-optimal (reverse) Horn fraction after renaming, $H_{\text{subopt}}(\mathcal{F}, \mathcal{R})$, is the (reverse) Horn fraction obtained after applying routine \mathcal{R} .*

For a maximization problem, the performance ratio of an algorithm is defined as the solution value found by the algorithm divided by the true optimal value. Boros [21] presents a rounding procedure for computing a sub-optimal Horn fraction after renaming that has a guaranteed performance ratio of $\frac{40}{67}$ on 3-SAT formulas. This procedure starts by solving a particular linear program. If the optimum value of this linear program is reasonably large, the rounding procedure uses the linear programming solution as input, otherwise the vector with all 0.5s is used to determine which variables have to be flipped.

2.2.2 Variants and extensions

Chandru and Hooker [26] proved \mathcal{NP} -hardness for the related problem of finding a renamable Horn subproblem with as many variables as possible in stead of maximizing the number of Horn clauses. Chandru and Hooker [25] extend the class of Horn clauses to a larger class of clauses for which the satisfiability problem can also be solved in linear time. The problem class by Chandru and Hooker [25] is extended by Schlipf, Annexstein, Franco and Swaminathan [126]. No polynomial time algorithm for recognizing these enlarged and extended Horn formulas is known up to now. Benoist and Hebrard [11] present a polynomial time algorithm for recognizing a special subclass of the extended and enlarged Horn formulas from Chandru and Hooker [25] and Schlipf et al. [126].

Porschen and Speckenmeyer [116] show that satisfiability restricted to mixed CNF-formulas (with a 2-SAT and a (hidden) Horn part) is \mathcal{NP} -complete (although both 2-SAT and Horn satisfiability can be solved in linear time). They also show that any CNF-formula can be transformed into such a mixed formula in polynomial time.

Nishimura, Ragde and Szeider [111] study the detection of so-called backdoor sets with respect to the polynomial time solvable classes of 2-SAT and Horn satisfiability. A strong backdoor set for a CNF formula is a set of variables S such that for each possible

assignment of the truth values to the variables in S the resulting simplified formula is solvable in polynomial time. A weak backdoor set W is a subset of the variables such that for a particular truth assignment to the variables in W , the resulting formula is solvable in polynomial time. The authors present an algorithm for detecting strong backdoor sets of at most a certain size k (if in existence) and show that it is hard to find weak backdoor sets.

2.3 Phase transitions

Phase transitions for the satisfiability problem are one of the main themes in Chapter 3. In that chapter, we present experimental evidence that random $\{3\}$ -SAT instances at the density threshold show a phase transition with respect to another parameter, the k -level reverse Horn fraction. In this section, we give a short overview of some of the literature on phase transitions.

It is well known that the class of random 3-SAT instances is subject to threshold phenomena with respect to the density parameter. Low-density problems are satisfiable with high probability and easily shown to be satisfiable with DLL techniques. High density problems are unsatisfiable with high probability. Instances with a low density tend to have many solutions, which makes it relatively easy to find one. Instances with a high density offer many different possibilities to prove a contradiction. Instances near the density threshold that are satisfiable have few solutions, and those that are not have few different ways to prove unsatisfiability which makes them hard to solve for satisfiability solvers.

For the satisfiability problem the two most important phase transitions are a phase transition with respect to satisfiability and a phase transition from easy to hard problems. A satisfiability phase transition with respect to a certain parameter p means that with low values of p there is a large majority of satisfiable (or unsatisfiable) instances. With increasing p the fraction of satisfiable (or unsatisfiable) instances decreases till all instances are unsatisfiable (satisfiable).

2.3.1 Random k -SAT

In most of the experiments in Chapter 3, random $\{3\}$ -SAT instances are considered. The OKgenerator by Kullmann [92] can randomly generate k -SAT instances with a specified number of variables and density. Many papers have been published about the phase transition between satisfiable and unsatisfiable instances with respect to the density. Friedgut [56] shows the existence of a function $d_0(n)$ for random 3-SAT such that the phase transition is within an ϵ -neighborhood of $d_0(n)$. In the report underlying [93] it is shown that $d_0(n) = 4.247$ is reasonably close to the phase transition for random 3-SAT instances with $n \leq 400$.

Franco [55] gives an overview of the history of results about threshold phenomena in satisfiability. Monasson et al. [108] and Biroli, Cocco and Monasson [14] study threshold phenomena with respect to the density parameter in random 3-SAT and the

class of random $(2 + p)$ -SAT problems. In a $(2 + p)$ -SAT formula, a fraction p of the clauses has three literals, while the others have two literals.

Goerdt [61] proved the existence of the threshold for 2-SAT at clause-variable density 1. However, there is no transition between easy and hard instances at this density. Coppersmith, Gamarnik, Hajiaghayi and Sorkin [31] prove that MAX-2-SAT shows a phase transition at density 1, just as 2-SAT. The detailed experimental results by Shen and Zhang [130] on the phase transition for MAX-2-SAT show that in contrast to 2-SAT, MAX-2-SAT shows a transition between easy and hard instances near the threshold.

2.3.2 Horn satisfiability

Horn clauses and threshold behavior play a central role in Chapter 3. Therefore, we will shortly mention some of the papers on phase transitions for Horn satisfiability although it is not directly related to this thesis. Informally, a threshold with respect to a certain parameter is sharp if the fraction of satisfiable instances as a function of the parameter is steep around the threshold and coarse otherwise.

Istrate [80] studies the phase transition in random Horn satisfiability from a probability theory point of view. He proves that for a class of Horn formulas made by selecting uniformly and independently m Horn clauses from the set of all possible Horn clauses, the threshold is coarse. Dunne and Bench-Capon [50] have proved a sharp threshold for k -Horn satisfiability for all fixed $k \geq 2$. Experiments by Demopoulos and Vardi [44] on randomly generated Horn satisfiability instances with only clauses of length 1 and 3 indicate the existence of a phase transition between a region where almost all CNF-formulas are satisfiable and a region where almost all are unsatisfiable. For Horn satisfiability restricted to instances with only clauses of length 1 and 2, no such phase transition has been observed.

2.3.3 Hard problem instances

Random k -SAT instances are most difficult at the density threshold. By selecting instances with a k -level reverse Horn fraction close to the phase transition, it is possible to generate instances that are significantly more difficult than average instances at that density. Generating hard instances is also one of the goals in Chapter 5.

Cook and Mitchell [30] point out that finding hard instances is an important issue in satisfiability research. Hard instances are useful both for understanding the computational complexity and for providing difficult benchmarks for comparing satisfiability solvers. Because relatively difficult instances are often encountered at, or in the neighborhood of the threshold, thresholds play an important role in generating difficult problem instances. Xu and Li [152] present new families of CNF-formulas that are hard for resolution and propose models with many hard instances and exact phase transitions.

Most generators for generating hard problem instances generate both satisfiable

and unsatisfiable instances. For testing incomplete satisfiability solvers, one needs a set of only satisfiable instances. Of course, it is possible to generate a mixed set of instances, solve them with a complete satisfiability solver and select only the satisfiable instances. However, this limits the classes and sizes of instances to the ones for which both satisfiable and unsatisfiable instances can be solved by a complete solver. Therefore, Achlioptas, Gomes, Kautz and Selman [1] propose a generator that only generates satisfiable instances with varying hardness that show an easy-hard-easy phase transition with respect to a certain parameter.

2.3.4 Phase transitions in some other problems

In this subsection, we mention three problems different from satisfiability that also show phase transition behavior in order to illustrate that phase transition behavior is not a special characteristic for the satisfiability problem but also occurs for other combinatorial problems.

Graph coloring

The density parameter is as strongly correlated to satisfiability for random 3-SAT instances, as is the vertex-edge density for 3-colorability of random graphs. Culberson and Gent [34] show that problems at the phase transition between colorable and uncolorable graphs are difficult to solve because of the absence of small and easily checkable subgraphs which can be used to quickly prove uncolorability. Svenson and Nordahl [136] experimentally study the phase transition for k -SAT and graph- k -coloring using Monte Carlo simulation.

Planning

Rintanen [122] presents two types of planning problem instances which show a phase transition. In the phase transition region, a planner based on satisfiability algorithms performs better than two planners using a heuristic local search procedure.

Quasigroup Completion problem

The Quasigroup Completion Problem is the problem of determining whether the empty entries in a partially filled Latin Square can be filled in such a way as to obtain a complete Latin Square. A Latin Square is an $N \times N$ -square that has to be filled with N times the numbers $1, \dots, N$ such that no number occurs more than once in a same row or column. From Gomes and Shmoys [63] and the references therein it is clear that the \mathcal{NP} -complete Quasigroup Completion problem has a phase transition from almost all solvable to almost all unsolvable instances with respect to the fraction of pre-assigned elements.

2.4 Maximum satisfiability

In Chapter 4, an algorithm based on semidefinite programming is presented that computes upper- and lower bounds on the number of clauses that can be satisfied. The problem of finding the maximal number of clauses that can be satisfied in a CNF-formula is called MAX-SAT. In this section, we give a very short overview of the literature on MAX-SAT, with a focus on approaches that use Operations Research Techniques.

MAX- k -SAT is the restriction of MAX-SAT to instances in which each clause has at most k literals. Of the MAX- k -SAT family, MAX-2-SAT and MAX-3-SAT are most widely studied. The importance of MAX-2-SAT lies in the fact, among others, that well-known \mathcal{NP} -hard problems like maximum cut and independent set can be reduced to MAX-2-SAT in an efficient way. Garey, Johnson and Stockmeyer [59] proved that MAX-2-SAT is \mathcal{NP} -hard. The literature on the approximation of MAX-2-SAT relevant for this thesis is mainly based on semidefinite programming and hence described in Subsection 2.7.

2.4.1 Approximation algorithms

The algorithm from Chapter 4 is an approximation method for MAX-SAT. In this section, we give a short overview of the literature on approximation algorithms for MAX-SAT.

Håstad [67] proved that for any $\epsilon > 0$ there does not exist a polynomial time $(\frac{7}{8} + \epsilon)$ -approximation algorithm for MAX-3-SAT unless $\mathcal{P} = \mathcal{NP}$. Trevisan, Sorkin, Sudan and Williamson [138] present an approximation algorithm for MAX-3-SAT with a performance guarantee of 0.801 by using so-called gadgets. Gadgets are finite combinatorial structures translating a given constraint of one optimization problem into a set of constraints of another optimization problem. For example a 3-SAT clause $X_1 \vee X_2 \vee X_3$ can be translated into the set of 2-SAT clauses

$$\begin{aligned} X_1, X_2, X_3, \neg X_1 \vee \neg X_2, \neg X_2 \vee \neg X_3, \neg X_3 \vee \neg X_1, \\ Y, X_1 \vee \neg Y, X_2 \vee \neg Y, X_3 \vee \neg Y. \end{aligned}$$

This performance guarantee is improved by Karloff and Zwick [84], who describe a polynomial time $\frac{7}{8}$ -approximation algorithm for MAX-3-SAT based on semidefinite programming, which will be discussed in more detail in Chapter 4. The proof in the paper by Karloff and Zwick is complete for satisfiable MAX-3-SAT instances but not for the unsatisfiable ones. Zwick [156] completes the proof by showing that the method is also a $\frac{7}{8}$ -approximation for unsatisfiable instances.

Dantsin, Gavrilovich, Hirsch and Konev [35] show how to construct an $(\alpha + \epsilon)$ -approximation algorithm for MAX-SAT from a given polynomial time α -approximation algorithm. The $(\alpha + \epsilon)$ -approximation algorithm runs in order $c^{\epsilon m}$ with c a constant dependent on α . This estimates the cost of improving the perfor-

mance ratio of an algorithm. They give similar constructions for MAX-2-SAT and MAX-3-SAT.

2.4.2 Algorithms based on integer linear programming

Joy, Mitchell and Borchers [82] present a branch-and-cut algorithm for MAX-SAT using two types of cuts: resolution cuts and odd cycle cuts. Resolution cuts combine two clauses such that a literal, say, X_i occurs in one of them and $\neg X_i$ occurs in the other. Odd cycle constraints combine cuts obtained earlier that contain a cycle. A cycle occurs for example if X_1 and X_2 occur in an inequality, X_2 and X_3 in another, X_3 and X_4 in a next and X_4 and X_1 in a next.

Borchers and Furman [20] describe an algorithm for optimally solving MAX-SAT instances and compare their approach with the approach by Joy, Mitchell and Borchers. Their algorithm uses GSAT [129] as a first phase to find a good approximation. The second phase is a DLL-like procedure. The algorithm of Borchers and Furman is faster on MAX-3-SAT problems, while the algorithm by Joy, Mitchell and Borchers is faster on MAX-2-SAT instances.

Alsinet, Manyá and Planes [3] present two branch-and-bound algorithms for MAX-SAT. These are variants on the algorithm by Borchers and Furman. They present experimental evidence that these two algorithms perform better on randomly generated MAX-2-SAT and MAX-3-SAT instances than the algorithm by Borchers and Furman [20].

Shen and Zhang [131] present three techniques that improve branch-and-bound algorithms for MAX-2-SAT. They developed a new lower bound and a preprocessing rule and variable ordering. Their experimental results show that their branch-and-bound algorithm performs better on many types of instances than the ones by Borchers and Furman [20] and Alsinet, Manyá and Planes [3].

2.4.3 Local search approaches

Stützle, Hoos and Roli [135] give an overview of the literature on local search algorithms for MAX-SAT. Mastrolilli and Gambardella [105] analyze the worst-case behavior of a tabu search algorithm on MAX-2-SAT as a function of the length of the tabu list. They show that tabu search has a better worst-case performance than basic local search if the length of the tabu list is linear in the number of variables in the CNF-formula. Porschen, Speckenmeyer, Randerath and Gärtner [117] present an experimental comparison of two local search methods, i.e. WalkSAT [128] and a tabu search method with dynamical tabu list length, on a class of CNF-formulas coming from level graphs. The idea of level graphs is to find a mapping of nodes in the graph to levels such that as few as possible arc crossings occur when the graph is plotted in the plane.

2.5 Graph-coloring

In this section, we give a very short overview of the literature on graph coloring. Graph coloring is one of the well-known problems that can be efficiently translated to satisfiability. The graph coloring problems is a subject in two chapters in this thesis. It is the main topic of Chapter 5. In that chapter, a method is presented for generating so-called forbidden color instances based on graph coloring instances. Experiments illustrate that these forbidden color instances are at least as hard as the graph coloring instances they originate from. In Chapter 3, the observed correlations are not only studied for random k -SAT problems but also for satisfiability instances coming from graph-3-coloring instances.

Graph- k -coloring is the problem whether a graph $G = (V, E)$ can be colored with at most k colors such that each vertex is colored and vertices connected by an edge have a different color. This problem is known to be \mathcal{NP} -hard. Guruswami and Khanna [65] show that coloring a 3-colorable graph with four colors is already an \mathcal{NP} -hard problem and that this even holds for bounded degree 3-colorable graphs.

Let $|V|$ be the number of vertices in G , and $|E|$ the number of edges. Two vertices i and j are called adjacent if they are connected by an edge. The *chromatic* number of a graph is the minimum number of colors that is needed to color all vertices of the graph such that no two adjacent vertices receive the same color.

An independent set is a set of vertices containing no adjacent vertices. Finding the minimal number of colors to color the graph is equivalent to partitioning the vertices in the graph into a minimal number of independent sets.

2.5.1 Satisfiability and graph coloring

In this section, we will mention a few papers combining satisfiability and graph coloring. Problems like 3-SAT and graph-3-coloring can be formulated as certain types of Constraint Satisfaction Problems (CSP). Beigel and Eppstein [9] present a fast algorithm for these so-called (3,2)-CSPs and based on this algorithm shows that graph-3-coloring can be solved in $\mathcal{O}(1.3289^{|V|})$ time.

Monasson [107] studies the behavior of DLL-like backtrack search algorithms for the graph coloring problem using ideas from statistical physics.

2.5.2 Approximation algorithms

Blum [16] studies polynomial-time approximation for coloring a graph known to be 3-colorable with as few colors as possible. He presents an algorithm that can color any 3-colorable graph with $\mathcal{O}(|V|^{3/8} \text{polylog}|V|)$ colors and extends the results to k -colorable graphs. Karger, Motwani and Sudan [83] present a randomized polynomial-time algorithm based on semidefinite programming that colors a 3-colorable graph with $\min\{\mathcal{O}(\Delta^{1/3} \log^{1/2} \Delta \log|V|), \mathcal{O}(|V|^{1/4} \log^{1/2}|V|)\}$ colors where Δ is the largest degree of a vertex. These results are extended to k -colorable graphs. Blum and Karger [17] show

how the results by Blum [16] and Karger, Motwani and Sudan [83] can be combined to find a coloring with $\tilde{O}(|V|^{3/4})$ colors for any 3-colorable graph, where \tilde{O} hides logarithmic factors. Paschos [115] gives an overview of polynomial-time approximation algorithms for graph coloring and improves the approximation ratio. Krivelevich and Sudakov [91] describe a randomized polynomial algorithm that colors almost every graph with $|V|/(\log_2|V| + c\sqrt{\log_2|V|})$ colors for each positive constant c . Marino and Dampier [102] present a genetic algorithm for graph coloring using symmetry breaking.

2.6 Linear Programming

In Chapter 3 a linear program forms an essential part of the algorithm aHS that is used for computing the k -level reverse Horn fractions, as defined in that chapter.

Let c be a row vector of length p , x a column vector of length p , b a column vector of length q and A a $q \times p$ -matrix. The standard formulation of a linear program is

$$\begin{aligned} & \text{maximize } cx && (2.6.1) \\ & \text{subject to } Ax \leq b \end{aligned}$$

x is a feasible solution of program (2.6.1) if it satisfies $Ax \leq b$, and it is an optimal solution if it attains the maximum. The best known and most used method to solve linear programs is the simplex method designed by Dantzig [36]. Linear programming can be solved in polynomial time, for example with the interior point method by Karmarkar [85]. Interior point methods for linear programming are widely studied as illustrated by the book of Roos, Terlaky and Vial [124]. This book describes the theory of linear optimization and gives an overview of many interior point methods for linear programming. CPLEX [79] provides a very efficient implementation of the simplex method, which we use in Chapter 3.

2.7 Semidefinite Programming and Satisfiability

Chapter 4 presents a method based on semidefinite programming for computing upper- and lower bounds on the number of clauses in a CNF-formula that can be satisfied. In this section, we describe some of the basics of semidefinite programming and give a short overview of the literature on semidefinite programming based algorithms for satisfiability and MAX-SAT.

2.7.1 Semidefinite programming in general

Semidefinite programming (SDP) is the problem of optimizing a linear function in a matrix variable X subject to linear constraints on the elements of X and the additional constraint that X must be a positive semidefinite matrix ($X \succeq 0$).

Definition 2.7.1. *A symmetric matrix A is defined to be positive semidefinite ($A \succeq 0$) if*

$$y^T A y = \sum_{i,j} A_{ij} y_i y_j \geq 0, \quad \forall y \in \mathbb{R}^n.$$

All eigenvalues of a positive semidefinite matrix are real and non-negative. Linear programming is a special case of semidefinite programming, where all matrices involved are diagonal. Semidefinite programs can be solved in polynomial time up to a given accuracy as can be concluded from the results of Nesterov and Nemirovskii [110]. For theory and algorithms on semidefinite programming we refer to the book of de Klerk [39]. Laurent and Rendl [94] give an overview of how semidefinite programming can be used for approximating combinatorial optimization problems.

2.7.2 Semidefinite programming methods for MAX- k -SAT

In their famous paper, Goemans and Williamson [60] introduced and analyzed an SDP-based approximation algorithm for MAX-2-SAT. Goemans and Williamson present both an SDP providing an upper bound for MAX-2-SAT and a randomized polynomial time rounding procedure for providing a lower bound. The performance guarantee for the approach by Goemans and Williamson is 0.878 for MAX-2-SAT. Based on this algorithm, a 0.7584-approximation algorithm for MAX-SAT is developed. Asano and Williamson [7] present a more extensive analysis of the performance guarantee of the algorithm by Goemans and Williamson for general MAX-SAT. By using the algorithms by Feige and Goemans [53] and Karloff and Zwick [84], a performance guarantee of 0.7846 can be obtained for MAX-SAT. Mahajan and Ramesh [100] present a method to derandomize this kind of randomized algorithms to obtain a polynomial-time deterministic algorithm giving the same approximation ratio as the randomized algorithm. Feige and Goemans [53] propose valid inequalities for this SDP and prove that with these inequalities, the performance guarantee increases to 0.931 for MAX-2-SAT. This performance guarantee for MAX-2-SAT is improved upon by Matuura and Matsui [106]. Their algorithm uses the SDP relaxation of Goemans and Williamson but a skewed distribution for the rounding procedure and realizes a performance guarantee of 0.935. To date, the best performance guarantee for MAX-2-SAT is obtained by the 0.940-approximation algorithm of Lewin, Livnat and Zwick [95] as far as we know. Håstad [67] proved that there is no $(\frac{21}{22} + \epsilon)$ -approximation algorithm for any $\epsilon > 0$ for MAX-2-SAT unless $\mathcal{P} = \mathcal{NP}$. Note that $\frac{21}{22} = 0.9545$, which means that the algorithm of Lewin, Livnat and Zwick is not far from being tight.

The approximation algorithm for MAX-3-SAT based on semidefinite programming by Karloff and Zwick [84] has a performance guarantee of $\frac{7}{8}$ for MAX-3-SAT. Because Håstad [67] showed that no approximation algorithm for MAX-3-SAT can achieve a performance guarantee better than $\frac{7}{8}$ unless $\mathcal{P} = \mathcal{NP}$, this implies that the algorithm by Karloff and Zwick is tight. Halperin and Zwick [66] present a semidefinite programming relaxation for MAX-4-SAT and a new class of rounding procedures. They seem to obtain a 0.8721-approximation algorithm for MAX-4-SAT.

De Klerk and Warners [42] show that the dual of the SDP relaxations for MAX-2-SAT can be solved efficiently with the solver DSDP [12]. They present a branch-and-cut algorithm for MAX-2-SAT that can solve instances with up to 50 variables and 500 clauses in a few minutes.

The semidefinite programming algorithms are not only of theoretical relevance but can also play a role in more practical applications, as is pointed out by Erlebach, Hall and Schank [51]. They use the algorithm of Lewin, Livnat and Zwick [95] for approximating the so-called MAXTOR problem. This problem implies finding an orientation of the edges in a directed graph G such that there is a maximum number of valid paths, i.e. paths not containing a vertex with two arcs pointing away from the vertex. This problem is relevant in classifying relationships on the Internet.

2.7.3 Proving unsatisfiability with semidefinite programming

De Klerk, van Maaren and Warners [41] introduce an algorithm for detecting unsatisfiability of CNF-formulas based on their elliptic approximation. The conditions for unsatisfiability are formulated as an eigenvalue optimization problem, which can be reformulated as a semidefinite program, the so-called gap relaxation. Their algorithm is complete on 2-SAT formulas and gives proofs of unsatisfiability in polynomial time for the pigeon hole and mutilated chess board problems. De Klerk and van Maaren [40] investigate the performance of the method from [41] on $(2+p)$ -SAT formulas, in which a fraction p of the clauses has three literals and all the others two. They give experimental results and performance guarantees.

Anjos [5] proposes a new SDP relaxation for satisfiability, which significantly improves on earlier SDP relaxations and can prove that a CNF-formula is unsatisfiable for formulas containing clauses of any length. Higher liftings are used to obtain the relaxation. The Gap relaxation [41] is the first lifting. In higher liftings, rows and columns are added that are indexed by subsets of variables additional to the rows and columns indexed by the variables. The experiments in the paper show that for instances with up to 260 variables and 400 clauses, satisfying assignments or proofs of unsatisfiability can be obtained. Anjos [4] presents an SDP relaxation for 3-SAT which is smaller than the one in [5] but can still detect satisfiability and unsatisfiability for 3-SAT instances.

2.7.4 Semidefinite programming algorithms

In the last decade, several semidefinite programming solvers have become available. Here, we will only shortly discuss some of the most effective and best-known ones.

- The solver **DSDP** [13] is a dual-scaling interior point algorithm which makes use of the structure and sparseness of the problem. Benson, Ye and Zhang [12] give the theoretical background of the algorithm and some computational results on MAX-CUT instances.

- **CSDP** ([19] and [18]) by Borchers is a predictor corrector variant of the algorithm designed by Helmberg, Rendl, Vanderbei and Wolkowicz [68]. One of the most important properties of this solver is that it is written in C. This makes the solver quite fast also partly caused by possibility to use the very efficient linear algebra routines in the packages LINPACK and LAPACK. CSDP is designed to handle sparse constraint matrices.
- The SDP-solver **SDPA** ([58], [153]) designed by Fujisawa, Kojima, Nakata and Yamashita is a primal-dual interior point algorithm based on the paper [57]. It is implemented in C++ using LAPACK for computations involving matrices. The solver is especially designed to work well for sparse matrices.
- **Sedumi** by Sturm [134] uses the self-dual embedding technique for solving semidefinite programs. It uses the sparsity structure of the program to decrease computation time. A worst-case iteration bound of $\mathcal{O}(\sqrt{n}\log(1/\epsilon))$ has theoretically been proven for Sedumi. A new version of SeDuMi was published by the Advanced Optimization Lab of McMaster University. The latest release of SeDuMi can be obtained from '<http://sedumi.mcmaster.ca/>'.

2.7.5 Sums of squares approximations

Hilbert's Positivstellensatz states that a non-negative polynomial in $\mathbb{R}[x_1, \dots, x_n]$ is a sum of squares in case $n = 1$, or has degree $d = 2$, or $n = 2$ and $d = 4$. Despite these restrictive constraints, explicit counter examples for the non-covered cases are rare, although Blekherman [15] proved that there must be many of them. Choi, Lam and Reznick [28] describe a class of polynomials that are non-negative but not a sum of squares.

Finding the minimum of a polynomial is an \mathcal{NP} -hard problem. A sufficient condition for a real-valued polynomial to be non-negative is the existence of a sum-of-squares decomposition. A lower bound on the minimum of a polynomial F on $\mathbb{R}[x_1, \dots, x_n]$ might be obtained by finding the largest ϵ such that $F - \epsilon$ is a sum of squares. This ϵ is an approximation, because not every polynomial that is non-negative is a sum of squares. A famous counterexample is the Motzkin-polynomial

$$M(x, y, z) = x^4y^2 + x^2y^4 + z^6 - 3x^2y^2z^2.$$

The computational experiments by Parrilo and Sturmfels [114] show that this ϵ is almost always equal to the true minimum for problems with up to 15 variables.

Parrilo [112], [113] shows that the existence of a sum-of-squares decomposition of a polynomial in \mathbb{R}^n can be verified by semidefinite programming methods. Prajna, Papachristodoulou, Seiler and Parrilo [119] present SOSTOOLS, a toolbox for solving sum of squares semidefinite programs. Waki, Kim, Kojima and Muramatsu [149] present a sum of squares relaxation that can be used to approximate the full SDP relaxation for polynomials with degree, say, k that contain relatively few of the possible monomials of degree at most k . Their computational experiments show that the

difference between the bounds obtained by their relaxation and the full relaxation is small for the problem instances they consider.

An approach for minimizing polynomials dual to a sum-of-squares approach is implemented in the package GloptiPoly [69]. This approach is based on the theory of moments and computes a number of lower bounds that monotonically converges to the global optimum. The advantage of this approach is that with the method described by Henrion and Lasserre [70] it provides a sufficient condition to check whether the exact optimal value is attained and a method to extract the corresponding solution.

Chapter 3

Correlations between reverse Horn fractions, satisfiability and solver performance

3.1 Introduction

In this chapter, we present extensive empirical evidence showing that an enhanced version of the reverse Horn fraction is correlated with the probability of fixed density random $\{3\}$ -SAT instances being satisfiable as well as with the computational effort of a variety of DLL-solvers in solving them. Horn clauses, reverse Horn clauses and reverse Horn fraction are defined in Subsection 2.2.1. This correlation even turns out to be observable for incomplete satisfiability solvers. This enhanced version of the reverse Horn fraction also turns out to effectively weakly separate 3-colorable from non-3-colorable instances for a class of graphs near threshold node-edge-density.

It is well-known that the class of random 3-SAT problems is subject to threshold phenomena with respect to the density parameter. Given a random 3-SAT instance, without unit clauses and two-literal clauses, its density d is defined as the ratio of the number of clauses and the number of variables. Low density problems are with high probability satisfiable, and easily shown to be satisfiable with DLL techniques. High density problems are with high probability unsatisfiable. However, for a sample of instances with equal density, a second parameter is necessary to (weakly) separate satisfiable and unsatisfiable instances.

In section 3.2 we introduce an enhanced version of the reverse Horn fraction, and present a greedy heuristic to compute it. The MAXSAT-fraction of a CNF-formula is defined as the maximal number of clauses that can be satisfied divided by the total number of clauses. The correlation between satisfiability and MAXSAT-fraction is obviously 100%. The formula is satisfiable if and only if the MAXSAT-fraction equals

This chapter is based on a paper published in the Annals of Mathematics and Operations Research [142] and one published in the proceedings of the SAT2003 conference [141]

1. This would suggest that a greedy approximation of this fraction is correlated to the satisfiability of the instance involved. Section 3.3 shows that this correlation is much weaker than the correlation with the enhanced reverse Horn fraction.

We observe a correlation between the enhanced reverse Horn fraction and computational effort of complete satisfiability solvers. In Section 3.4 we present evidence for the fact that instances with high enhanced reverse Horn fraction are easier to solve in case they are satisfiable, and more difficult to solve in case they are unsatisfiable, and vice versa. We selected a number of state-of-the-art solvers to illuminate this feature. In Section 3.5 we give evidence for the fact that the correlation between the enhanced reverse Horn fraction and computational effort of complete solvers extends to two state-of-the-art incomplete satisfiability solvers.

In section 3.6 we present large size experiments which support the fact that the correlation between reverse Horn fraction and satisfiability for random $\{3\}$ -SAT instances scales with respect to size, and is not just a small size artifact. In section 3.7 we study the correlation between reverse Horn fraction and satisfiability for satisfiability instances coming from graph-3-coloring instances. The algorithm aHS for calculating the enhanced reverse Horn fraction is described in Section 3.8. The conclusions are presented in Section 3.9.

3.2 Horn fractions

In this section, we introduce our enhanced reverse Horn fraction. Renaming, or flipping, a variable X_i means that every occurrence of X_i is replaced by $\neg X_i$, and every $\neg X_i$ is replaced by X_i . By renaming, the reverse Horn fraction of a CNF-formula might change.

Definition 3.2.1. *Given a CNF-formula \mathcal{F} and a deterministic renaming routine \mathcal{R} , the sub-optimal reverse Horn fraction after renaming, $H_{subopt}(\mathcal{F}, \mathcal{R})$, is the reverse Horn fraction obtained after applying routine \mathcal{R} .*

The algorithm aHS, described in Section 3.8, uses a deterministic greedy local search procedure to compute a sub-optimal reverse Horn fraction. This procedure is described in subsection 3.2.2. It is simple and its performance on random $\{3\}$ -SAT instances seems to be quite good, as is illustrated in Section 3.3.1. A search tree is a tree with the root node containing the empty assignment and nodes at deeper levels containing partial assignments. When going a level deeper in the search tree, the search space is split in disjoint parts and in each part at least one of the free variables is assigned a value. A leaf of the tree contains a full assignment or a partial assignment that can be proved to give a conflict.

Definition 3.2.2. *Given a CNF-formula \mathcal{F} , a deterministic renaming routine \mathcal{R} , and a deterministic search tree T which solves \mathcal{F} , the sub-optimal average reverse Horn fraction over the search tree (in short average reverse Horn fraction), $AH_{so}(\mathcal{F}, \mathcal{R}, T)$, is the average of the $H_{subopt}(\mathcal{F}_t, \mathcal{R})$ over all nodes t in the search tree T , with \mathcal{F}_t the CNF-formula at node t of T .*

To compute its average reverse Horn fraction, an instance has to be solved completely. For the next variant of reverse Horn fraction this is not necessary.

Definition 3.2.3. *Given a CNF-formula \mathcal{F} , a deterministic renaming routine \mathcal{R} , and a deterministic search tree T which solves \mathcal{F} , the k -level average sub-optimal reverse Horn fraction, $AH_{so}(\mathcal{F}, \mathcal{R}, T, k)$, is the average reverse Horn fraction over the nodes in T up to depth k .*

In this chapter \mathcal{R} is the routine described in subsection 3.2.2 and the algorithm aHS from Section 3.8 generates the search tree T involved. This k -level average sub-optimal reverse Horn fraction is the enhanced reverse Horn fraction mentioned in the introduction of this chapter. In the remainder, we write k -level reverse Horn fraction for k -level average sub-optimal reverse Horn fraction. The above definitions are sound because the renaming routine and search tree we use are deterministic. The renaming routine and search tree of aHS guarantee that the k -level reverse Horn fraction can be computed in polynomial time for fixed k . As a tool for (weakly) separating satisfiable and unsatisfiable instances appropriate choices for k depend on the size and nature of the instances. Dealing with instances of the size we investigated in this paper, $k = 1$ or $k = 2$ seem to be appropriate choices to reveal the correlations mentioned.

3.2.1 Weighted and MAX-Horn variants

Any of the above notions can be extended to weighted variants. In these variants, the weighted fraction of reverse Horn clauses is optimized or approximated. This weighted fraction is defined as

$$\frac{\sum_{l=1}^n w_l H_l}{\sum_{l=1}^n w_l S_l},$$

in which n is the number of variables, w_l is a weight of clauses of length l , S_l is the number of clauses of length l , and H_l is the number of reverse Horn clauses of length l . Varying weights might influence the observed distributions significantly. Experiments showed that this is the case for random 3-coloring instances but not for random $\{3\}$ -SAT instances.

One might consider also the k -level max-reverse-Horn fraction which is defined as the maximum of the reverse Horn fractions in the nodes of the search tree up till level k . These variants are used in the experiments on graph-3-coloring in Section 3.7.

Figure 3.1 shows the importance of choosing appropriate weights. We selected a set of forbidden color instances obtained as described in Chapter 5 based on a graph coloring instance called 'games'. In the figure we plotted with the solid line the difference between the average reverse Horn fraction μ_s of the satisfiable, and the average μ_u of the unsatisfiable instances. The dashed line gives the difference between the average max-reverse-Horn fraction of the satisfiable (max_s) and unsatisfiable (max_u) instances. From the figure it is clear that the difference between μ_s and μ_u and the difference between max_s and max_u is highly influenced by the chosen weight. With larger weights these differences are larger. As a consequence the correlation between

weighted reverse Horn fraction and satisfiability is stronger. Figure 3.1 also shows that the max-reverse-Horn fraction shows a stronger separation than the average reverse Horn fraction for these instances.

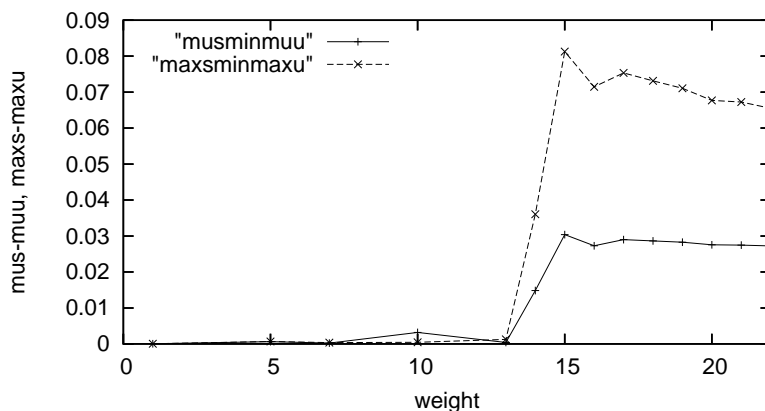


Figure 3.1: Games instance with different weights

We tested different weights on random graph-3-coloring instances with 200 nodes and 456 edges. The results with respect to the 1-level reverse Horn fraction are presented in Table 3.1. The unit and two-literal clauses get weight 1, and the 3-literal clauses get weight w_3 . The first column gives the weights w_3 , the second column the average μ_s of the 1-level reverse Horn fractions of the satisfiable instances, the third one gives the average μ_u of the 1-level reverse Horn fractions of the unsatisfiable instances. The column labelled d_{av} gives $\mu_s - \mu_u$, the one labelled max_s gives the average of the 1-level max-reverse-Horn fractions for the satisfiable instances, and max_u for the unsatisfiable ones. d_{max} is the difference $max_s - max_u$. The overlap is a measure for the quality of the separation between satisfiable and unsatisfiable instances. The smaller the overlap the better the separation. O_A is defined as the overlap with respect to the average reverse Horn fraction. The column labelled C_H gives the 1-level reverse Horn fraction for which about 50% of the instances is satisfiable. For larger 1-level reverse Horn fractions, there are more satisfiable instances. For smaller 1-level reverse Horn fractions, there is a majority of unsatisfiable instances. O_M is the overlap with respect to the 1-level max-reverse Horn fraction, C_{MH} gives the 1-level max-reverse Horn fraction with about 50% of the instances being satisfiable.

3.2.2 Greedy Horn heuristic

The optimal reverse Horn fraction is approximated by a deterministic greedy local search heuristic. This heuristic takes as input a CNF-formula and takes the following steps:

w_3	μ_s	μ_u	d_{av}	max_s	max_u	d_{max}	O_A	C_H	O_M	C_{MH}
1	0.9251	0.9259	-0.0007	0.9269	0.9269	0.0000	99.2%	0.9309	97.1%	0.931596
6	0.8387	0.8371	0.0015	0.8618	0.8498	0.0121	89.2%	0.8378	76.4%	0.8524
7	0.8451	0.8428	0.0022	0.8679	0.8559	0.0119	78.0%	0.8418	71.1%	0.8579
8	0.8531	0.8508	0.0023	0.8756	0.8639	0.0117	71.1%	0.8518	69.7%	0.8663
9	0.8617	0.8592	0.0024	0.8831	0.8719	0.0112	69.1%	0.8602	70.5%	0.8740
10	0.8696	0.8671	0.0024	0.8899	0.8789	0.0109	65.1%	0.8687	69.0%	0.8810
15	0.8989	0.8970	0.0018	0.9138	0.9061	0.0077	66.3%	0.8984	70.5%	0.9077
20	0.9174	0.9160	0.0014	0.9293	0.9232	0.0061	66.5%	0.9172	70.2%	0.9245

Table 3.1: Graph-3-coloring with different weights

Algorithm 2: Greedy Horn heuristic

- (1) Compute the reverse Horn fraction of the CNF-formula.
- (2) Compute for every variable X_i the resulting increase in reverse Horn fraction σ_i in case the variable would be flipped.
- (3) Determine $\sigma = \max_i \sigma_i$
- (4) **if** $\sigma > 0$ and $\sigma_i = \sigma$,
- (5) flip X_i and go to step 1.
- (6) **else if** $\sigma = 0$,
- (7) compute for every X_i with $\sigma_i = 0$, the increase ρ_i in the total number of positive literals. Determine $\rho = \max \rho_i$.
- (8) **if** $\rho > 0$ and $\rho_i = \rho$,
- (9) flip variable X_i and go to step 1.
- (10) **else**
- (11) STOP.

If several variables yield the same increase in reverse Horn fraction or number of positive literals, the variable with the smallest index number is chosen.

The algorithm for computing the optimal non-negative fraction, defined and used in Section 3.3, is analogous. In the algorithm above, one needs to replace only reverse Horn fraction by non-negative fraction.

3.3 Reverse Horn fraction and non-negative fraction

In this section, we motivate why we study k -level reverse Horn fraction as a parameter for weakly separating satisfiable from unsatisfiable instances. One could ask whether, for instance, a greedy MAXSAT-approximation could serve this purpose equally well, because an optimal MAXSAT-procedure of course shows complete separation. First, we present some arguments for studying MAXSAT-approximation for separation. Then, we present convincing experimental evidence showing that the correlation between k -level reverse Horn fraction and satisfiability is stronger than the correlation between a greedy approximation of the MAXSAT-fraction and satisfiability.

ity. In order to obtain an impression of the influence of the density parameter, we vary the density over the set of instances involved in the experiment. Finally, we illustrate that the correlation of k -level reverse Horn fraction and satisfiability gets stronger with increasing k .

3.3.1 Optimal reverse Horn fraction and MAXSAT-fraction

In this subsection, we present the concept of non-negative fraction, and illustrate why this might be an appropriate alternative for the reverse Horn fraction.

Definition 3.3.1. *A clause is non-negative if it contains at least one positive literal. The non-negative fraction of a CNF-formula is the number of non-negative clauses divided by the total number of clauses.*

The optimal non-negative fraction is the non-negative fraction after applying an optimal renaming procedure. The k -level non-negative fraction is defined analogous to the k -level reverse Horn fraction (see Definition 3.2.3).

The MAXSAT-fraction is equal to the optimal non-negative fraction. In a CNF-formula with a non-negative fraction p , one can satisfy p clauses by setting all variables to true. The other way around, a formula with MAXSAT-fraction p can be turned into a formula with non-negative fraction p by renaming all variables that are false in the MAXSAT-solution. For a $\{2\}$ -SAT-formula, containing only clauses with two literals, the non-negative fraction and the reverse Horn fraction are equal by definition. Because finding the optimal reverse Horn fraction and finding the MAXSAT-fraction are the same for 2-SAT, and MAX-2-SAT is \mathcal{NP} -hard as is proven by Garey, Johnson and Stockmeyer [59], the known fact that finding the optimal reverse Horn fraction is \mathcal{NP} -hard follows (see for a different proof of this fact [32]).

In reverse Horn clauses at most one literal is negative. In a non-negative clause at least one literal must be positive. For $\{2\}$ -SAT CNF formula, reverse Horn and non-negativeness are equivalent. Every reverse Horn clause with two or more literals is a non-negative clause, but for clauses with three or more literals the reverse is not true. Hence, non-negativeness is a less stringent constraint for clauses with at least three literals.

Because the non-negative fraction and the reverse Horn fraction are equal for $\{2\}$ -SAT, and because the optimal non-negative fraction is, in contrast to optimal reverse Horn fraction, directly related to satisfiability, the k -level non-negative fraction might be expected to weakly separate at least as good as the k -level reverse Horn fraction. However, our experiments show that the k -level reverse Horn fraction gives a better separation. One explanation might be that a reverse Horn clause stays reverse Horn if shortened in a search tree, while this is not true for non-negative clauses.

The greedy algorithms used to compute sub-optimal reverse Horn fraction and sub-optimal non-negative fraction are discussed in Section 3.2.2. In fact, both algorithms only differ in the notion that is maximized. Because the algorithms are essentially the

n	μ_s	μ_u
150	0.7102	0.7031
200	0.7108	0.7040
300	0.7106	0.7044
400	0.7105	0.7059

Table 3.2: The sub-optimal reverse Horn fraction for instances with increasing size, density 4.25

same, the difference in approximation algorithm does not play a role in the difference in separation possibilities.

To give an impression on the quality of the approximations, we computed our sub-optimal reverse Horn fraction and the sub-optimal non-negative fraction for 130 instances with 40 variables and density 4.25. For these instances, we computed the optimal reverse Horn fraction with the integer linear program suggested by Boros [21] and the MAXSAT-fraction with the complete MAXSAT-solver by Borchers and Furman [20]. The sub-optimal reverse Horn fraction is on average about 1.9% less than the optimal reverse Horn fraction. For the sub-optimal non-negative fraction and the MAXSAT-fraction, the difference is on average about 2.26%. This implies that the MAXSAT-fraction is slightly more difficult to approximate by our greedy sub-optimization procedure. We realize these instances are quite small compared to the instances in the remainder of this chapter. Therefore, we compared the sub-optimal non-negative fraction and the MAXSAT-fraction also for 800 instances with 100 variables and density 4.25. In that case the difference between the sub-optimal and the optimal value is on average 2.28%, which is comparable to the result for the instances with 40 variables. Computing optimal reverse Horn fractions for the instances with 100 variables is not possible in reasonable time with the method used. The sub-optimal reverse Horn fraction does not decrease with size as is shown in Table 3.2. In this table, μ_s is the average sub-optimal reverse Horn fraction of the satisfiable instances, and μ_u is the average of the unsatisfiable instances. This might indicate that the performance of our greedy heuristic does not deteriorate with increasing size.

The better separation possibilities for the k -level reverse Horn fraction compared to k -level non-negative fraction in section 3.3.2 might be partly caused by the better approximation, but we do not believe this plays an important role because the difference in error percentages is not that large. The difference between the average MAXSAT-fraction of satisfiable and unsatisfiable instances is 0.0062 for the instances with 40 variables and density 4.25. For optimal reverse Horn fraction, the difference between these two averages is about twice as large (0.013). This could explain why the separation is better for k -level reverse Horn fraction than for k -level non-negative fraction. More details about this experiment comparing reverse Horn and non-negative fraction are given in Table 3.3. In this table, NN indicates non-negative fraction, 'Horn' represents reverse Horn fraction. μ_s is the average sub-optimal fraction of the

	μ_s	σ_s	μ_u	σ_u	d_{av}	$\hat{\mu}_s$	$\hat{\sigma}_s$	$\hat{\mu}_u$	$\hat{\sigma}_u$	\hat{d}_{av}	ϵ
Horn	0.7151	0.0021	0.6979	0.0221	0.0172	0.7271	0.0175	0.7141	0.0158	0.0130	1.9%
NN	0.9772	0.0103	0.9716	0.0095	0.0056	1	0	0.9938	0.0012	0.0062	2.26%

Table 3.3: 40 variables, comparison of reverse Horn and non-negative fraction

satisfiable instances, μ_u the average sub-optimal fraction of the unsatisfiable instances. σ_s and σ_u are the corresponding standard deviations. d_{av} is the difference $\mu_s - \mu_u$. The parameters with a hat are the corresponding concepts with respect to the average optimal fraction. ϵ is the average difference between sub-optimal and optimal fractions.

3.3.2 Experiments on reverse Horn fraction and non-negative fraction

In this subsection, we will present experimental evidence that the k -level reverse Horn fraction has better separating possibilities than the k -level non-negative fraction. We used a set of random $\{3\}$ -SAT instances with 400 variables and densities 4.0, 4.1, 4.2, 4.3, 4.4 and 4.5, 2000 instances of each density.

Definition 3.3.2. *The cumulative distribution function $FH_s(x)$ of the satisfiable instances is defined as the probability that a satisfiable instance has a k -level reverse Horn fraction smaller than x .*

Definition 3.3.3. *The probability density function $fH_s(x)$ of the satisfiable instances is the derivative of $FH_s(x)$.*

Analogously, $FN_s(x)$ and $fN_s(x)$ are defined for the non-negative fraction. FH_u , FN_u , fH_u and fN_u are defined analogously for the unsatisfiable instances.

To compare the quality of the weak separation, we introduce the concept *overlap*. $FH_s(x)$ and $FH_u(x)$ are approximated by the fraction of instances with a reverse Horn fraction smaller than x . $FN_s(x)$ and $FN_u(x)$ are approximated by the fraction of instances with a non-negative fraction smaller than x . These are approximated with spline functions and differentiated to get fH_s , fH_u , fN_s and fN_u . C_H is defined as the point where the density functions of satisfiable and unsatisfiable instances cross.

The overlap equals the area below corresponding density functions of both satisfiable and unsatisfiable instances. The first picture in Figure 3.2 shows the fH_s and fH_u with respect to the 0-level and 2-level reverse Horn fraction, the second one for the 1-level reverse Horn fraction. The solid curves are the f_s , and the dashed curves the f_u . The left two are f_u and f_s with respect to the 0-level reverse Horn fraction and the right-hand side pair are f_s and f_u with respect to the 2-level reverse Horn fraction. Figure 3.2 shows that the k -level reverse Horn fractions are larger with search tree depth $k = 2$ than with $k = 0$. In Figure 3.2 the overlap is the shaded area. Figure 3.3 gives the probability density functions for the 0- and 1-level non-negative

fraction. From the pictures can be seen that the overlap is considerably larger for the non-negative fraction than for the reverse Horn fraction.

The overlap can be computed without using the probability density functions, only using the approximation of the probability distributions, which are numerically more stable than their derivatives. To see this, note that the point where the density functions cross is exactly the point where the difference between both cumulative distributions is minimal. The smaller the overlap, the better the separation.

In Table 3.4, we give detailed results for the densities 4.2, 4.3 and 4.4. For instances with density 4.0, 4.1 and 4.5, it is not meaningful to compute the overlap. With density 4.0 there is only 1 unsatisfiable instance, for density 4.1 only 54 unsatisfiable instances, and for density 4.5 there are only 10 satisfiable instances. Table 3.5 gives results for the complete set of instances with densities 4.0, 4.1, 4.2, 4.3, 4.4 and 4.5. Detailed results for $k = 0, 1$ and 2 for the k -level reverse Horn fraction and for the non-negative fraction for $k = 0$ and $k = 1$ are given. In Tables 3.4 and 3.5 the average k -level reverse Horn (or non-negative) fraction for both the satisfiable (μ_s) and unsatisfiable (μ_u) instances are given, and their difference $d_{av} = \mu_s - \mu_u$. Columns σ_s and σ_u indicate respectively the standard deviations. O_H is the overlap and C_H the fraction where the densities functions of satisfiable and unsatisfiable instances cross.

d	k	μ_s	σ_s	μ_u	σ_u	d_{av}	O_H	C_H
4.2	0	0.7109	0.0067	0.7058	0.0063	0.0051	0.674	0.709
	1	0.7396	0.0073	0.7331	0.0068	0.0065	0.628	0.737
	2	0.7526	0.0066	0.7454	0.0058	0.0072	0.559	0.748
4.3	0	0.7105	0.0065	0.7055	0.0063	0.0050	0.672	0.708
	1	0.7388	0.0067	0.7324	0.0068	0.0064	0.642	0.735
	2	0.7511	0.0061	0.7443	0.0060	0.0068	0.568	0.749
4.4	0	0.7103	0.0056	0.7047	0.0068	0.0056	0.638	0.705
	1	0.7378	0.0064	0.7308	0.0071	0.0070	0.593	0.735
	2	0.7498	0.0054	0.7422	0.0063	0.0076	0.500	0.746

Table 3.4: 400 variables, densities 4.2, 4.3, and 4.4

	n	k	μ_s	σ_s	μ_u	σ_u	d_{av}	O_H	C_H
Horn	400	0	0.7122	0.0070	0.7044	0.0067	0.0078	0.571	0.708
		1	0.7420	0.0082	0.7305	0.0072	0.0115	0.446	0.736
		2	0.7557	0.0081	0.7419	0.0067	0.0138	0.344	0.749
NN	400	0	0.9772	0.0033	0.9751	0.0033	0.0021	0.732	0.977
		1	0.9800	0.0030	0.9774	0.0030	0.0026	0.665	0.979

Table 3.5: 400 variables, density 4.0, 4.1, 4.2, 4.3, 4.4 and 4.5, k -level reverse Horn and non-negative fraction

For 0-level reverse Horn fraction, the overlap O_H measured over the test set turns out to be 0.571. For 0-level non-negative fraction, the overlap is 0.732. Hence, the correlation between 0-level reverse Horn fraction and satisfiability turns out to be stronger than for 0-level non-negative fraction. Around the fraction C_H , where the

probability density functions f_s and f_u cross, instances with any of the densities occur in comparable numbers, which shows that the differences in distributions of satisfiable and unsatisfiable instances are not due to density influences only but are essentially a result of the parameter reverse Horn fraction itself.

The 1-level reverse Horn fraction improves on the results for the 0-level reverse Horn fraction; the overlap is only 0.446. For the 1-level non-negative fraction, the overlap is 0.665. Hence, also on level 1 the reverse Horn fraction performs better than the non-negative fraction. For the 2-level reverse Horn fraction, the overlap decreases to 0.344.

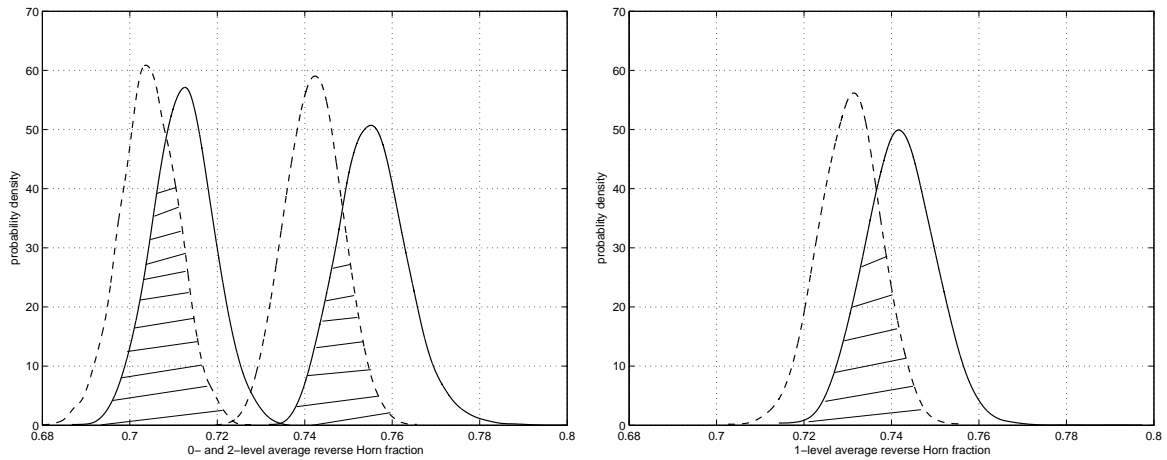


Figure 3.2: 400 variables, densities 4.0 to 4.5 joined, probability density functions for 0-, 1- and 2-level reverse Horn fraction

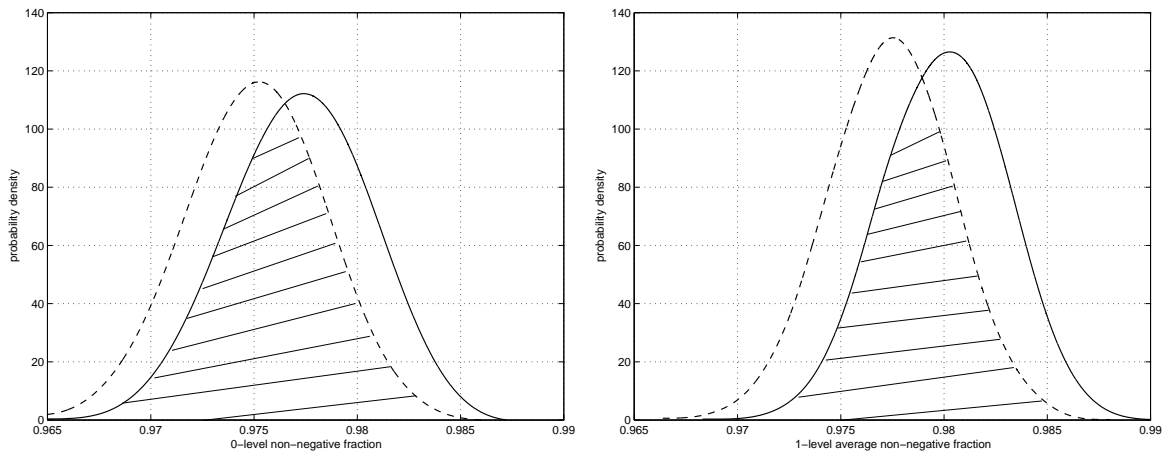


Figure 3.3: 400 variables, densities 4.0 to 4.5 joined, probability density functions for 0- and 1-level non-negative fraction

Note that these instances with varying densities are easier to separate than instances with a fixed density. For fixed density instances with 400 variables and density

4.25, we show in section 3.6 that the overlap with respect to the 0-level reverse Horn fraction and 1-level reverse Horn fraction are respectively 0.718 and 0.672.

3.4 Correlation with computational effort

In this section we present empirical evidence for the fact that k -level reverse Horn fraction and computational effort are correlated for a variety of DLL solvers. To exclude influences from the density parameter, all random $\{3\}$ -SAT instances in these experiments have density 4.25, a density generally believed to be close to the threshold density for the sizes involved. First, we estimate the average computational effort of the selected solvers, in order to present the results independent of size. We give an illustration why scaling with the estimated average computational effort is suitable, and show the difference in the distributions of satisfiable and unsatisfiable instances. We conclude the section with empirical evidence for the correlation between k -level reverse Horn fraction and computational effort for the different solvers. For convenience, we will denote number of nodes by *run time*.

The selected complete satisfiability solvers are the OKsolver [93], knfs [45] and aHS. The first two are selected because of their good performance on random CNF-formulas. aHS is selected because its explicit use of reverse Horn fractions. knfs and OKsolver are briefly described in Chapter 1. The solver aHS is described in Section 3.8. aHS is a complete solver with the special property that the search tree is not binary. aHS is our tool to compute k -level reverse Horn fractions if it is run up to level k , but aHS is also a complete satisfiability solver if it is run till completion.

3.4.1 Estimation of average run time

We will briefly describe how the average run time can be estimated. Estimating the average run time for each solver and size is necessary to be able to present size independent results for the correlation between k -level reverse Horn fraction and run time. The *scaled run time* of an instance with a given size is defined as the run time of the instance divided by the estimated average run time of that particular size.

For estimating the average run time, we used 13,000 random $\{3\}$ -SAT instances with the number of variables between 125 and 400. These instances are solved by the OKsolver and knfs and the smaller instances among those by aHS. For any of the selected sizes, this results in the average run time for that particular size. For each solver, we compute an exponential fit to these data points ((number of variables, average run time)), which estimates the average run time. In Figure 3.4 the exponential fits estimating run time are plotted. The horizontal axis gives the number of variables, the vertical axis the estimated average run time. The upmost curve is the average run time for aHS, the middle one is for the OKsolver and the lowest for knfs.

Each exponential fit is of the form

$$RT_{av} = \alpha e^{\beta n},$$

with RT_{av} the average run time, and n the number of variables. The parameters α and β are given in Table 3.6 for each solver. The exponential fits constitute a good approximation of the data. The maximum gap between the function value of the fit and the observed data points for the OKsolver is 2.3%. For knfs the maximum gap is 2.7% and for aHS 4.2%.

solver	α	β
OKsolver	1.1877	0.0328
knfs	0.2864	0.0325
aHS	0.0652	0.0610

Table 3.6: Parameters of exponential fit for the three solvers

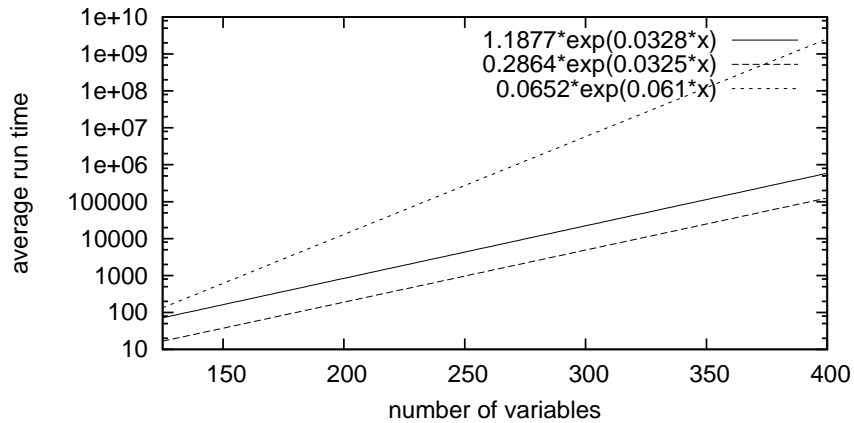


Figure 3.4: Average run times for OKsolver, knfs and aHS

3.4.2 Scaled run time distribution

In this subsection, we show that the estimation made above works well in scaling run times for different sizes. We observe that the probability distributions of run times of satisfiable and unsatisfiable instances have a different shape.

Figure 3.5 shows the probability distributions with respect to the scaled run time for the OKsolver for instances with 150 to 400 variables. On the horizontal axis is indicated the scaled run time. On the vertical axis is the fraction $F(t)$ of instances solved before run time t . The logarithmic-shaped curves concern satisfiable instances and the S-shape curves the unsatisfiable instances. The names of the curves are given in top-down order in the figure. We conclude that the scaled run time distribution is quite size independent. Similar pictures emerged for the other solvers.

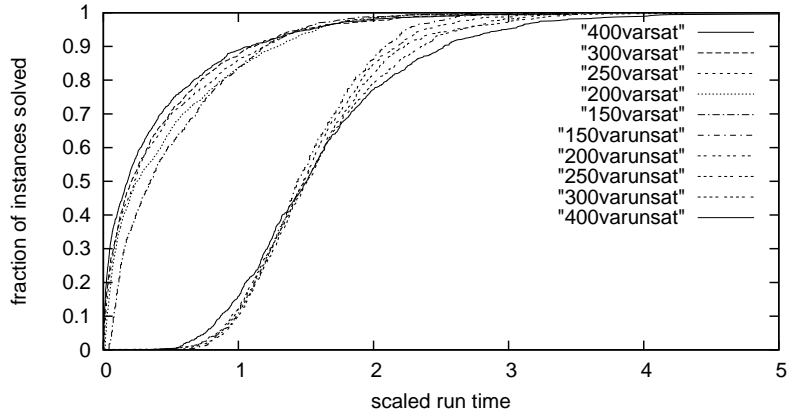


Figure 3.5: Scaled run time distribution, OKsolver, 150 to 400 variables

3.4.3 Correlation between reverse Horn fraction and run time

Now we show how k -level reverse Horn fraction and run time are correlated for the fixed-density instances considered. For the sizes considered, we take $k = 2$ to illustrate these features. Figure 3.6 shows a global trend in the correlation between k -level reverse Horn fraction and run time for OKsolver, knfs and aHS.

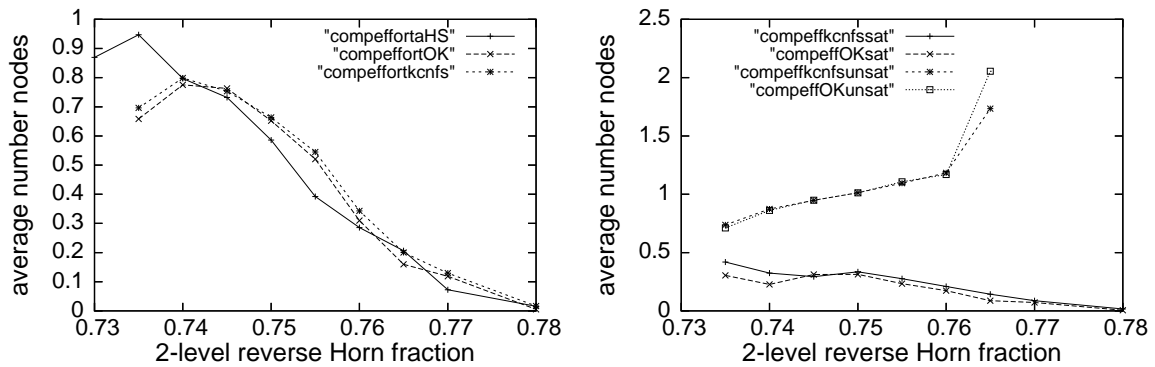


Figure 3.6: 400 variables, $d=4.25$, OKsolver and knfs and 150 variables, $d=4.25$, aHS

In the first picture in Figure 3.6 we indicate on the vertical axis the average number of nodes in the search tree created by the OKsolver, knfs and aHS, scaled with the average number of nodes it took the solver concerned to solve the unsatisfiable instances of the indicated size. On the horizontal axis is indicated the 2-level reverse Horn fraction computed with aHS. The instances concerned are 2000 random $\{3\}$ -SAT instances with 400 variables, for the OKsolver and knfs, and 300 instances of 150 variables for aHS. All instances have density 4.25 and are generated by the OKgenerator [92]. Notice that instances of 400 variables are difficult to solve or even unsolvable in reasonable time for most solvers. The second figure in Figure 3.6 represents the

average number of nodes for the satisfiable and unsatisfiable instances separately for the OKsolver and kcnfs.

The figure shows that there is an obvious correlation between the size of the search tree and the 2-level reverse Horn fraction established using aHS. From the first figure it is clear that for all of the three solvers the average size of the search tree shows an increase/decrease pattern peaking in the vicinity of the fraction C_H where the probability density functions cross. From the second figure it can be concluded that satisfiable instances get easier when the 2-level reverse Horn fraction increases while unsatisfiable instances contrarily get more difficult to solve.

Satisfiable instances get on average easier when k -level reverse Horn fraction increases, and unsatisfiable instances get more difficult to solve with increasing k -level reverse Horn fraction. The intuition behind satisfiable instances with a small k -level reverse Horn fraction being more difficult is that these instances probably have only a few satisfying assignments. On the other hand, unsatisfiable instances with a large k -level reverse Horn fraction are difficult because a large reverse Horn fraction indicates few conflicts in the formula. Hence, it is difficult to find these conflicts to prove unsatisfiability.

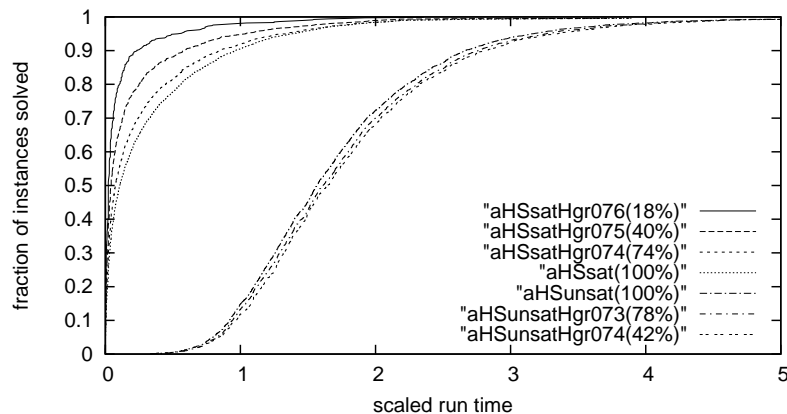


Figure 3.7: aHS, 125, 150, 175 and 200 variables, density 4.25

The instances used in Figure 3.7 are random $\{3\}$ -SAT instances with 125, 150, 175 and 200 variables. On the horizontal axis is indicated the scaled run time, on the vertical axis the fraction of instances $F(t)$ that is solved before run time t . The names of the curves are in the top-down order of the curves. The lowest logarithmic-shaped curve corresponds to the complete set of satisfiable instances. If we restrict the set of satisfiable instances to the instances with a 2-level reverse Horn fraction larger than or equal to 0.74, 0.75 and 0.76 respectively, we observe that the curves become steeper. The percentages given in Figure 3.7 are the percentages of instances that respect the corresponding lower bound on the 2-level reverse Horn fraction. The difference between the curves indicates that satisfiable instances with a larger 2-level reverse Horn fraction are easier for aHS to solve. Similarly, we plotted in Figure 3.7 three curves for the

unsatisfiable instances, the upmost S-shaped curve for the complete set of unsatisfiable instances, the second one for instances with a 2-level reverse Horn fraction larger than 0.73 and the lowest one for unsatisfiable instances with a 2-level reverse Horn fraction larger than 0.74. The order of the curves illustrates that contrarily to the satisfiable instances, unsatisfiable instances with a larger 2-level reverse Horn fraction are more difficult to solve.

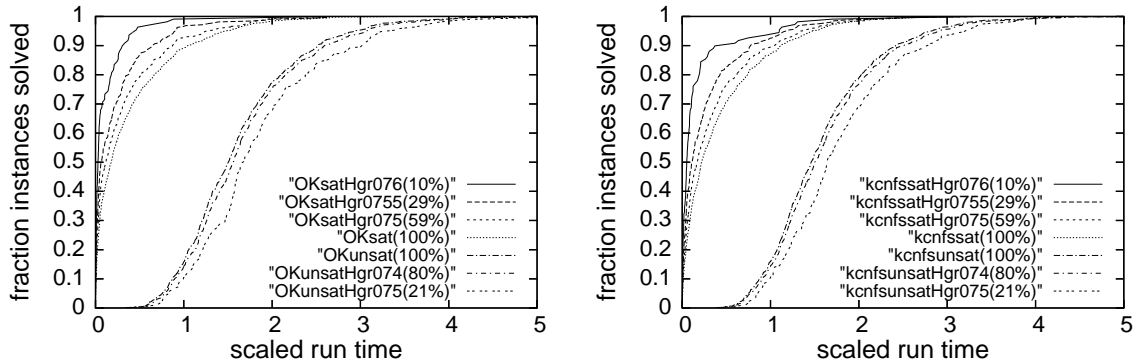


Figure 3.8: OKsolver and kcnfs, 400 variables, density 4.25

Similar pictures emerge for the OKsolver and kcnfs, also for larger instances (with 400 variables), as can be seen in Figure 3.8. We observe a same shape as in Figure 3.7. We conclude that the correlation between 2-level reverse Horn fraction and run time for the OKsolver and kcnfs is well observable.

3.5 Correlation between reverse Horn fraction and performance of incomplete solvers

In this section, we study the correlation between the k -level reverse Horn fraction and the performance of incomplete solvers. For this study, we selected two state-of-the-art incomplete solvers: SP [22] and UnitWalk [74]. Both solvers are briefly described in Chapter 1.

SP is designed for really large instances (more than 10,000 variables). Hence, the performance on the small instances with 400 variables in our experiment is not representative for the performance of SP. However, we decided to include the results because they still give some insight in the correlation between k -level reverse Horn fraction and solver performance.

3.5.1 SurveyPropagation

This subsection shows a correlation between 2-level reverse Horn fraction and the success rate of SP, in the sense that SP finds a solution more often for instances with a larger 2-level reverse Horn fraction than for ones with a smaller 2-level reverse

Horn fraction. We ran SP on 1000 satisfiable instances with 400 variables and density 4.25. On the complete set of instances SP finds a solution for 38.8% of the instances. However, if we partition the instances in six groups of about the same number of instances with increasing 2-level reverse Horn fraction, the success rate of finding a solution increases from 26.9% in the first group (with lowest 2-level reverse Horn fractions) to 68.8% in the last group. Table 3.7 gives the detailed results for this experiment. It shows a clear correlation between 2-level reverse Horn fraction and the performance of SP.

2-level reverse Horn fraction	Success rate
[0,0.745]	26.9%
(0.745,0.75]	29.6%
(0.75,0.755]	38.0%
(0.755,0.76]	45.7%
(0.76,1]	68.8%

Table 3.7: SurveyPropagation on instances with 400 variables and density 4.25

3.5.2 UnitWalk

For UnitWalk it turned out impossible to analyze its performance on the same set of instances as we did for SP, because the success rate on the same benchmark set is 100%. (We will make use of this excellent performance in Section 3.6.) To get an impression of the influence of reverse Horn fraction on UnitWalk’s performance, we let it try to solve 500 instances with 600 variables and density 4.25. These instances are currently too large to be solved in reasonable time with a complete solver. For 59.4% of the instances UnitWalk finds a solution within 600 seconds (clock time). For these proven satisfiable instances, a figure similar to Figure 3.7 showed up (not included because of its similarity). Differences between the curves are somewhat less pronounced than for OKsolver, kcnfs and aHS.

3.6 Correlation between reverse Horn fraction and satisfiability

In this section we will examine the correlation between reverse Horn fraction and satisfiability for random $\{3\}$ -SAT instances with the number of variables between 150 and 600 and the density equal to 4.25. We show that the correlation between k -level reverse Horn fraction and satisfiability exists for instances with up to 600 variables. For each size, we use 1000 satisfiable and 1000 unsatisfiable instances. The instances are all close to the density threshold. The instances with up to 400 variables are solved with a complete satisfiability solver. Solving the instances with 600 variables with a

3.6. CORRELATION BETWEEN REVERSE HORN FRACTION AND SATISFIABILITY 47

complete solver takes too much time. Later in this section, we explain how the results for the instances with 600 variables are obtained.

Based on the density alone, an instance has a probability of being satisfiable of about 50%. For any of the instances, we computed the reverse Horn fraction of the formula in the root node (level 0), the 1-level and 2-level reverse Horn fraction. From these data we computed the average k -level reverse Horn fraction for both the satisfiable (column μ_s) and unsatisfiable (column μ_u) instances, and their difference $d_{av} = \mu_s - \mu_u$. d_{av} is an indication of the quality of the separation between the distributions of the k -level reverse Horn fraction of satisfiable and unsatisfiable instances. Columns σ_s and σ_u indicate respectively the involved standard deviations. For each size and level, the fraction C_H where both probability density functions cross is given. At this point the probability of an instance being satisfiable is 0.5. The overlap O_H is given in the table.

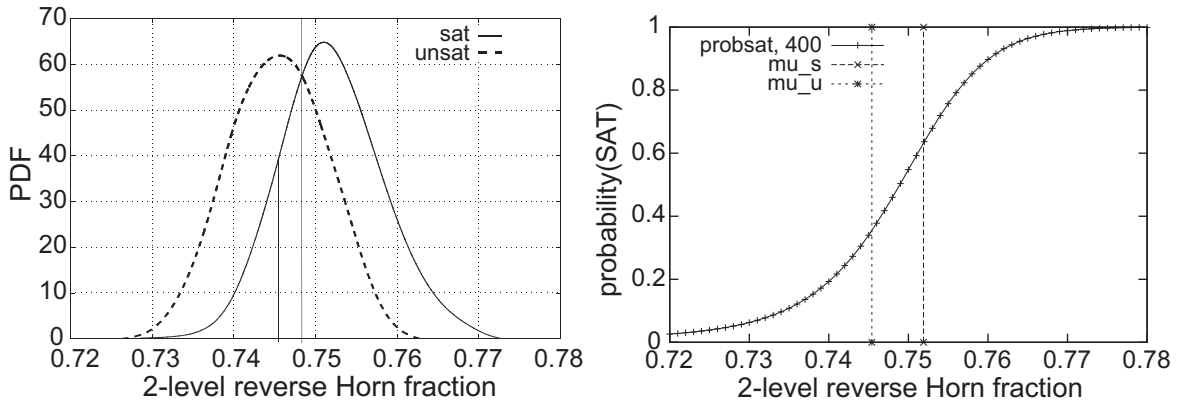


Figure 3.9: 400 variables, probability density and probability of being satisfiable

From Table 3.8, we can conclude various trends. First, for fixed k , the difference $\mu_s - \mu_u$ decreases, but the overlap O_H shows relatively stable. Further, for fixed size, the overlap decreases, and hence, the separation improves with increasing k . To illustrate this somewhat more we added $k = 3$ for the 400 variable instances.

The first picture in Figure 3.9 shows the probability density functions of satisfiable and unsatisfiable instances with 400 variables and density 4.25 over the 2-level reverse Horn fraction. C_H and μ_u are indicated by the two vertical lines. The second picture gives for each value of the 2-level reverse Horn fraction the probability that an instance with the corresponding fraction is satisfiable. The vertical lines indicate μ_s and μ_u . As mentioned, for the complete set of instances the probability of being satisfiable is about 50%, but, for example, for instances with a 2-level reverse Horn fraction larger than 0.756, this probability of being satisfiable is 80%.

We use the excellent performance of UnitWalk to obtain the results for the instances with 600 variables. We ran UnitWalk on 2000 instances with 600 variables and density 4.25 with a limit on (clock) time of 1800 seconds. UnitWalk finds a solution for 1149 instances (57.5%). Because the density is close to the 3-SAT threshold it is likely

n	k	μ_s	σ_s	μ_u	σ_u	d_{av}	O_H	C_H
150	0	0.7102	0.0113	0.7031	0.0112	0.0071	0.720	0.708
	1	0.7381	0.0164	0.7260	0.0108	0.0121	0.64	0.733
	2	0.7489	0.0155	0.7349	0.0099	0.0140	0.576	0.743
200	0	0.7108	0.0094	0.7040	0.0095	0.0068	0.700	0.706
	1	0.7378	0.0107	0.7287	0.0094	0.0091	0.643	0.7344
	2	0.7489	0.0102	0.7383	0.0085	0.0106	0.566	0.7446
250	0	0.7102	0.0086	0.7044	0.0082	0.0058	0.730	0.7078
	1	0.7378	0.0092	0.7302	0.0084	0.0076	0.663	0.7347
	2	0.7493	0.0085	0.7409	0.0074	0.0084	0.593	0.7458
300	0	0.7106	0.0078	0.7050	0.0077	0.0056	0.713	0.7079
	1	0.7385	0.0083	0.7315	0.0081	0.0070	0.67	0.7557
	2	0.7506	0.0078	0.7428	0.0072	0.0078	0.601	0.7471
350	0	0.7105	0.0069	0.7051	0.0067	0.0054	0.692	0.708
	1	0.7388	0.0074	0.7321	0.0073	0.0067	0.650	0.735
	2	0.7512	0.0072	0.7438	0.0067	0.0074	0.591	0.7478
400	0	0.7105	0.0065	0.7059	0.0063	0.0046	0.718	0.7084
	1	0.7390	0.0069	0.7333	0.0066	0.0057	0.672	0.7364
	2	0.7519	0.0063	0.7454	0.0059	0.0065	0.591	0.7489
	3	0.7585	0.0061	0.7517	0.0056	0.0068	0.586	0.755
600	0	0.7102	0.0055	0.7060	0.0054	0.0042	0.664	0.709
	1	0.7392	0.0063	0.7338	0.0060	0.0054	0.622	0.737
	2	0.7530	0.0053	0.7473	0.0050	0.0057	0.558	0.750

Table 3.8: Random {3}-SAT from 150 to 600 variables

that most of the remaining instances are unsatisfiable. We tested UnitWalk also on 500 instances of this size and density 4.27. From these instances, UnitWalk finds a solution in 44.6% of the cases. A second indication that at most for only a few satisfiable instances a solution is not found is the fact that the average 0-level reverse Horn fraction for the instances with 600 variables does not deviate much from that for the instances with 400 variables for which is sure that all unsatisfiable instances are really unsatisfiable. A satisfiable instance that is not solved by UnitWalk is likely to have a 0-level reverse Horn fraction smaller than the average, because the satisfiable instances with a small 0-level reverse Horn fraction tend to be difficult to solve. If many instances would not be solved by UnitWalk the average 0-level reverse Horn fraction would be considerably higher for the satisfiable instances with 600 variables than for the ones with 400 variables, which is not the case. From these facts, we conclude that in order to establish empirical evidence for the correlation between satisfiability and k -level reverse Horn fraction, one might read 'satisfiable' as 'solved by UnitWalk' and 'unsatisfiable' as 'not solved by UnitWalk' rather safely: the small number of errors made presumably will not essentially alter the probability density functions measured.

Table 3.8 gives the details on the distribution of the k -level reverse Horn fraction for instances with 600 variables. Behavior with respect to expected 1- and 2-level reverse Horn fraction, standard deviations and overlap is according to the trend for the smaller instances with up to 400 variables. Figure 3.10 illustrates this trend. We plotted the average 2-level reverse Horn fraction for both the satisfiable and unsatisfiable instances as function of size. Below any of these lines, we indicate the 25%-percentile, i.e. that fraction such that 75% of the instances has a larger 2-level reverse Horn fraction. Analogously, 75%-percentiles are indicated. Expected 2-level reverse Horn fractions μ_s and μ_u increase with size. It shows that the correlation between 2-level reverse Horn fraction and satisfiability does not get weaker with increasing size. This fact advocates that these correlations are not 'small size artifacts'.

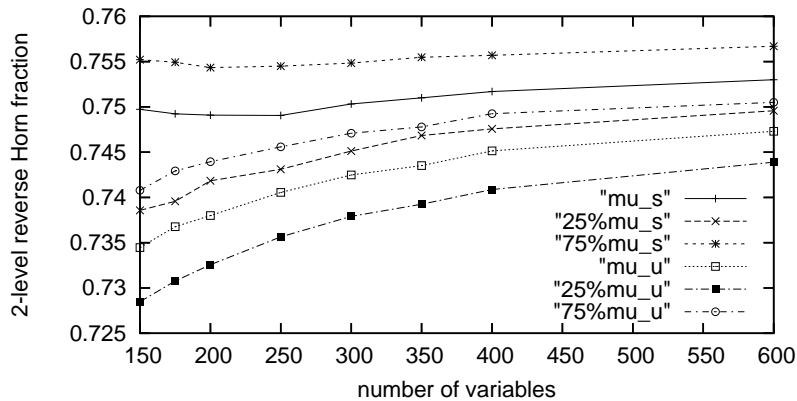


Figure 3.10: Expected 2-level reverse Horn fraction

3.7 Graph-3-coloring

To investigate whether the correlations observed are typical for random $\{3\}$ -SAT we ran several experiments with graph 3-coloring instances, coded as satisfiability problems. The problem is to check whether the nodes in a graph can be colored with three colors such that no two adjacent nodes have the same color. Let N be the number of nodes in the graph, and E the number of edges. To encode the graph-3-coloring problem as a satisfiability problem variables X_{kc} are introduced being true if node k has color c , and false otherwise. The encoding of the graph 3-coloring problem has N clauses of length three, expressing that every node has to be assigned one of the three colors, $3N$ clauses of length two formulating for each node and pair of colors that the node does not have both colors, called unicity clauses, and finally there are $3E$ clauses of length two ensuring for every edge that both end points do not have the same color.

For these graph-3-coloring instances, the k -level reverse Horn fraction can predict quite well whether an instance is satisfiable or not, but the optimal reverse Horn fraction in the root node does not give any hint, because the latter is equal for every instance of a certain size. By renaming all variables, all clauses of length two have two positive literals, and the clauses of length three have three negative literals. Hence, the number of reverse Horn clauses, after this renaming, is $3N + 3E$, independent of the particular instance. We cannot improve on this, because to get one of the clauses of length three reverse Horn, we need to rename at least two variables. These two variables occur together in one of the $3N$ unicity clauses, so one of the clauses with two literals ceases to be reverse Horn.

Besides the version of the problem described above, we also considered a variant without unicity clauses. These clauses are not strictly necessary to find a feasible coloring of the nodes, because we can always choose one of the colors a node gets if it is multiple-colored. This variant turns out to be easier, and the fact that all instances have the same reverse Horn fraction at the root node does not hold, in absence of the unicity clauses.

We experimented with the weight of the clauses. It turned out that the weight has a significant influence on both computation times and the shape of the distribution. This influence turned out to be more prominent for the problem without unicity clauses, but it is also present for the problem with unicity clauses. Since the CNF-formulas have roughly seven times more two-literal clauses as three-literal ones, we chose the weighted variant where 'being reverse Horn' of a three-literal clause is considered seven times more important than 'being reverse Horn' of a two-literal clause. We confirmed experimentally that this weight indeed showed the best separation properties.

N	E	k	μ_s	σ_s	μ_u	σ_u	d_{av}	O_H	C_H
100	222	1	0.8674	0.0204	0.8440	0.0046	0.0234	0.321	0.852
		2	0.9029	0.0248	0.8491	0.0053	0.0538	0.069	0.860
150	333	1	0.8523	0.0102	0.8445	0.0039	0.0078	0.587	0.848
		2	0.8662	0.0150	0.8487	0.0041	0.0175	0.261	0.854
		3	0.8882	0.0253	0.8532	0.0042	0.0350	0.121	0.861
150	339	1	0.8503	0.0109	0.8431	0.0040	0.0072	0.656	0.849
		2	0.8649	0.0180	0.8483	0.0041	0.0166	0.378	0.854
		3	0.8813	0.0261	0.8505	0.0043	0.0308	0.164	0.857

Table 3.9: Graph-3-coloring with respect to the k -level reverse Horn fraction

We generated 500 instances with 100 nodes and 222 edges. For this density, the probability that an instance is satisfiable is approximately 50%. We generated also 600 instances with 150 nodes with 333 edges, i.e. with the same density as the instances with 100 nodes, but also with 339 edges, seemingly closer to the threshold density for this size. The results are given in Table 3.9, which illustrates that the correlation between k -level reverse Horn fraction and satisfiability is much more prominent than

for random $\{3\}$ -SAT instances. Considering the k -level max-reverse Horn fraction, we even get better results, as is shown in Table 3.10.

N	E	k	μ_s	σ_s	μ_u	σ_u	d_{av}	O_H	C_H
150	333	1	0.8985	0.0536	0.8555	0.0075	0.0430	0.524	0.864
		2	0.9809	0.0353	0.8650	0.0083	0.1159	0.085	0.887
		3	0.9989	0.0085	0.8690	0.0084	0.1299	0.002	0.900
150	339	1	0.8931	0.0542	0.8549	0.0077	0.0382	0.554	0.865
		2	0.9688	0.0492	0.8635	0.0079	0.1053	0.182	0.881
		3	0.9967	0.0125	0.8682	0.0084	0.1285	0.009	0.892

Table 3.10: Graph-3-coloring with respect to k -level max-reverse Horn fraction

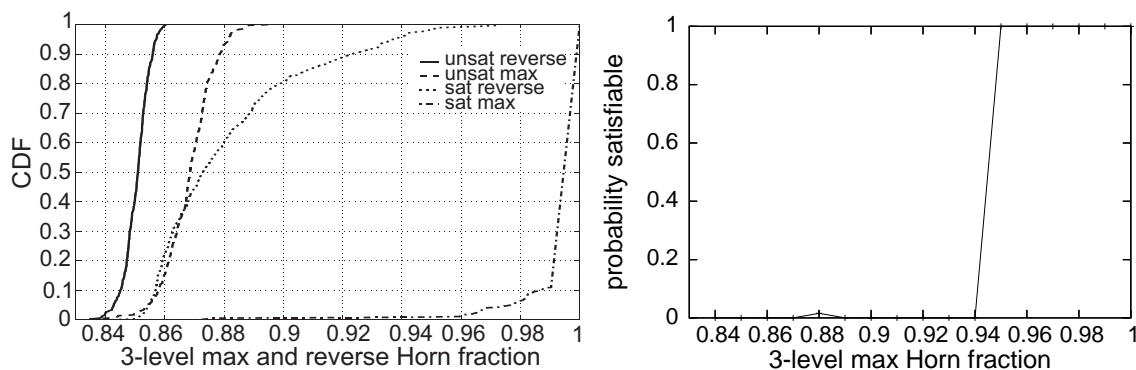


Figure 3.11: Graph-coloring, 150 nodes, 339 edges, cumulative distribution functions

In Figure 3.11 the cumulative distribution functions for the satisfiable and unsatisfiable instances with 150 nodes and 339 edges are shown, both with respect to the 3-level reverse Horn fraction and the 3-level max-reverse Horn fraction. On the vertical axis are the distribution functions, indicating the probability that an instance has a 3-level reverse Horn or max-reverse Horn fraction smaller than x . These distributions illustrate that the correlation between satisfiability and 3-level max-reverse-Horn fraction is more prominent (almost complete separation over the sample) than for 3-level reverse Horn. The phenomena show much more convincing than for random $\{3\}$ -SAT. The second picture in Figure 3.11 gives the probability that an instance is satisfiable as a function of the 3-level max-reverse-Horn fraction.

Let w_3 be the weight of the three-literal clauses, and $w_2 = 1$ the weight of the two-literal clauses. The first picture in Figure 3.12 gives in case $w_3 = 1$ and $w_3 = 4$ the probability that an instance is satisfiable as a function of the 2-level reverse Horn fraction for a set of randomly generated graph-3-coloring instances formulated with unicity clauses. The second picture gives the same type of figure for the same set of instances in a formulation without unicity clauses.

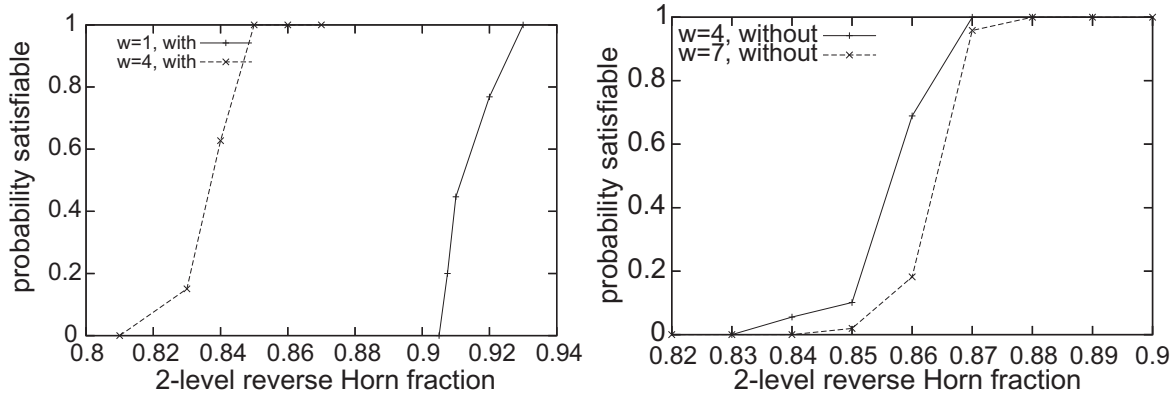


Figure 3.12: Graph-coloring, 100 nodes, 222 edges, probability being satisfiable

3.8 The algorithm aHS

Besides the algorithm aHS, we present in this section some statistics about a typical run on a $\{3\}$ -SAT instance. Also, some aspects related to the choice of the objective function of the linear programming relaxation are discussed. First, we describe the main steps of the algorithm aHS. Steps 2, 4 and 5 are described in more detail below. We emphasize that aHS is not designed to be an efficient satisfiability solver, but to generate many nodes at lower levels to make the averaging process more reliable.

- (1) Identify all unit clauses; propagate the corresponding values through the CNF-formula until no unit clauses remain.
- (2) Apply failed literal tests (FLT) in order to simplify the formula as much as possible. If the formula is proved unsatisfiable or a satisfying assignment is found: STOP. Otherwise: step 3.
- (3) Greedy suboptimize the reverse Horn fraction using the greedy local search renaming heuristic as described in subsection 3.2.2.
- (4) Solve an associated linear program. Let $\omega = (\omega_1, \dots, \omega_n)$ be a solution (splitting point) to this linear program.
- (5) Define $n + 1$ subproblems using a decomposition of the search space based on ω .
- (6) Go into breadth-first recursion.

The failed literal tests are explained in subsection 3.8.1. The associated linear programming relaxation is described in subsection 3.8.2. Subsection 3.8.3 clarifies the branching strategy from step 5. In subsection 3.8.4 a justification of the linear program is given together with an example.

3.8.1 Failed literal tests

Iterative unit propagation is the process of propagating the literals in unit clauses till no unit clauses remain. In the failed literal tests, every variable is set temporarily to true. If this leads to a contradiction after iterative unit propagation the variable can be fixed to false for the current node and all of its children. This value is propagated followed by iterative unit propagation. If this leads to a contradiction we can conclude that the formula is unsatisfiable and backtrack.

These failed literal tests are continued until none of the variables can be concluded to be false in a full loop over the variables. After this loop, an analogous loop for fixing the variables to false is made. We repeat the process until none of the variables could be fixed during the last loop over the positive and negative assignments.

3.8.2 Linear Program

Let n be the number of variables, and m the number of clauses of a satisfiability instance. As is common knowledge, an instance of the satisfiability problem can be described by a binary integer linear program. Let A be a $m \times n$ 0-1 matrix, with $a_{ij} = 1$ if variable X_j is contained in clause i , $a_{ij} = -1$ if $\neg X_j$ is contained in clause i , and 0 otherwise. Let x be a 0-1 vector with $x_j = 1$ if X_j is set to true, and 0 otherwise. For the vector b of right hand sides, b_i equals 1 minus the number of negated variables in clause i . With this notation a satisfying assignment x satisfies

$$\{Ax \geq b, x \in \{0, 1\}^n\}.$$

To illustrate this formulation the clause $X_p \vee \neg X_q \vee X_r$ gives the linear constraint $x_p - x_q + x_r \geq 0$.

The constraint $x \in \{0, 1\}^n$ is replaced by $0 \leq x_i \leq 1, i = 1, \dots, n$ to obtain the associated linear programming relaxation in step 4. The objective of the linear program is to maximize $\sum x_j$. Setting all free variable to 0.5 satisfies all linear inequalities. With the objective function we increase the chance that the solution to the linear programming relaxation is integer-valued. If an integer-valued linear programming solution is found, this is a solution to the integer linear program (and hence the CNF-formula), and the algorithm can be stopped.

3.8.3 Search tree design

The search tree design described in this subsection is based on [140]. The authors of this paper show that integer programs having only inequalities satisfying the so-called generalized Horn condition

$$\alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_n x_n \geq \alpha,$$

with at most one α_i negative, can be solved effectively for feasibility using an homotopy procedure and a certain subdivision of Euclidean space. This subdivision now is precisely what we use in our splitting of the search space.

The region G_i is the cone spanned by the vectors from

$$E_i = \{-e_0, -e_1, \dots, -e_n\} \setminus \{-e_i\},$$

in which e_j is the unit vector with a 1 in the j -th row, e_0 is the vector with ones in all entries, and e_i is one of the vectors from $\{e_0, e_1, \dots, e_n\}$. The regions that follow from this splitting are described below.

If the formula is not solved in a search tree node, $n + 1$ subproblems are defined using a decomposition of the search space based on the linear programming solution ω . The first region G_0 is defined as:

$$G_0 = \{x = (x_1, \dots, x_n) \mid x_i \leq \lfloor \omega_i \rfloor, \text{ for all } i\}.$$

Each G_i , $i = 1, \dots, n$, is defined by

$$x_i \geq \lfloor \omega_i \rfloor + 1, \quad (3.8.1)$$

$$x_i - x_k \geq \lfloor \omega_i - \omega_k \rfloor + 1, \quad k < i, \quad (3.8.2)$$

$$x_i - x_k \geq \lceil \omega_i - \omega_k \rceil, \quad k > i. \quad (3.8.3)$$

$\lfloor x \rfloor$ is defined as the largest integer smaller than or equal to x . $\lceil x \rceil$ is the smallest integer larger than or equal to x . Clearly, the region is empty if $\omega_i = 1$. Otherwise, we have $x_i = 1$ or X_i is true. For the case $k > i$, we have $\neg X_k$ if $\lceil \omega_i - \omega_k \rceil = 1$. For the case $k < i$, we have $\neg X_k$ if $\lfloor \omega_i - \omega_k \rfloor = 0$, and we can conclude that the region is empty if it is 1.

The branching strategy results in fact only in adding a number of unit clauses to the formula in any of the regions G_0 to G_n . Assume that we ordered the boolean variables X_1, \dots, X_n in non-decreasing order of the corresponding solution $(\omega_1, \dots, \omega_n)$ of the linear programming solution. Let $X_{[i]}$ be the i -th variable in this ordering. If the solution does not have all ones, region $G_{[1]}$ is simply obtained by fixing $X_{[1]}$ to 1. All other (free) variables remain free. In general, $G_{[i]}$ for each i such that $X_{[i]}$ is not fixed, and $\omega_{[i]} < 1$ is as follows:

$$G_{[i]} = \{x \mid x_{[i]} = 1, x_{[k]} = 0 \text{ for each } k < i\}.$$

By constraint (3.8.1), region i is empty if $\omega_{[i]} = 1$. In each G_i these region constraints are propagated before FLT is applied and the linear programming relaxation in the node solved.

Example To illustrate this approach, take, for example, $\omega = (1, \frac{3}{4}, \frac{3}{4}, 0)$. In this case the regions are

$$G_0 = \{x_1 \leq 1, x_2 = x_3 = x_4 = 0\},$$

$$G_1 = \emptyset,$$

$$G_2 = \{x_2 = 1, x_4 = 0, x_1, x_3 \text{ free}\},$$

$$G_3 = \{x_3 = 1, x_2 = x_4 = 0, x_1 \text{ free}\},$$

$$G_4 = \{x_4 = 1, x_1, x_2, x_3 \text{ free}\}.$$

To see that G_1 is empty note that $\omega_1 = 1$. $x_1 \geq \lfloor \omega_1 \rfloor + 1$ implies $x_1 \geq 2$. For example, for G_2 , $x_4 = 0$ follows from $\lceil \omega_2 - \omega_4 \rceil = 1$, hence $x_2 - x_4 \geq 1$. Combining with $x_2 = 1$ this gives $x_4 = 0$. Note that G_4 is much larger than the other regions.

In the two dimensional space the splitting of the search space graphically looks as follows

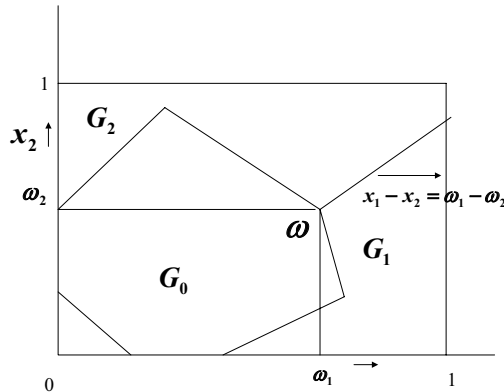


Figure 3.13: Splitting the search space into regions

Notice that ω is determined by the only two inequalities not satisfying the generalized Horn condition. This is the main intuition behind the splitting: try to eliminate many non-Horn clauses by splitting the search space at ω . Notice that in higher dimensions ω will be determined by more inequalities being not generalized Horn.

3.8.4 The objective of the linear program

The objective of the linear program is to maximize $\sum x_j$. We give a justification for using this objective function. The main intuition behind the search space splitting is: try to eliminate many non-reverse Horn clauses. By eliminating non-reverse Horn clauses the CNF-formula is supposed to get closer to a reverse Horn formula, which hopefully makes it easier to (partially) solve by the failed literal tests. A linear inequality is defined to be stringent to the optimal linear programming solution if the slack of this inequality is 0 with respect to this solution. The slack of the linear inequality $ax \geq b$ is $ax - b$. A clause is defined to be *stringent* if the corresponding linear inequality is stringent to the optimal linear programming solution. A clause is defined to be *affected* in region G_i at level 1 if it is either satisfied or shortened in region G_i . The objective function tries to get as many as possible non-reverse Horn clauses stringent to the optimal linear programming solution. These stringent clauses turn out to have a larger probability to be affected at the next level in the search tree.

To illustrate this, we use a two-literal non-reverse Horn clause, but one could give similar arguments for non-reverse Horn clauses with more literals. Assume without loss of generality $\omega_p \leq \omega_q$. A clause $\neg X_p \vee \neg X_q$ in the formula is satisfied at the next

objective value	156.5
$\frac{1}{n} \sum_i \omega_i$	0.78
% of stringent clauses	27%
% of stringent non-reverse Horn clauses	75%
fraction of regions not empty after propagating (1)-(3)	8.5%
fraction of regions not empty and not solved after FLT	6.0%
average % of clauses affected	16.4%
average % of stringent clauses affected	19.2%
average % of affected stringent clauses that are non-reverse Horn	61.5%

Table 3.11: Statistics for an instance with 200 variables

level in all regions G_i with $\omega_i > \omega_p$ if $i < p$ and $\omega_i \geq \omega_p$ if $i > p$, because of the definitions of the regions. Now, for a *stringent* clause of the type $\neg X_p \vee \neg X_q$, it holds that $(1 - \omega_p) + (1 - \omega_q) = 1$, resulting in $\omega_q = 1 - \omega_p$. Hence, the clause is satisfied in each region G_i with $\omega_i > \min(\omega_p, \omega_q) = \min(\omega_p, 1 - \omega_p)$, which is always smaller than or equal to 0.5. The linear program maximizes $\sum x_j$ and $(x_1, \dots, x_n) = (0.5, \dots, 0.5)$ is feasible for the linear program. Hence, $\frac{1}{n} \sum_i \omega_i$ is at least 0.5. However, for random $\{3\}$ -SAT CNF-formulas, the average ω_i of the root-node linear program turns out to be about 0.75. This implies that a stringent clause $\neg X_p \vee \neg X_q$ is very likely to be satisfied, because most of the ω_i -values will be larger than $\min(\omega_p, 1 - \omega_p)$. From the above, we might conclude that stringent non-reverse Horn clauses are satisfied in most nodes at the next level.

Example The following example illustrates the above. We consider an arbitrary selected random $\{3\}$ -SAT instance with 200 variables and density 4.25. For this instance, we solved the linear programming relaxation in the root node and examined the nodes at level 1. The average percentage of clauses affected is the average over the percentages of affected clauses in the non-empty regions G_i at level 1. Analogously, the average percentage of stringent clauses affected and the average percentage of affected stringent clauses that are not-reverse Horn are defined. The statistics of this instance are presented in Table 3.11.

We see that for the root-node linear program 75% of the stringent clauses are non-reverse Horn, which means that non-reverse Horn clauses are more likely to be stringent than reverse Horn clauses. At level 1 stringent clauses are more likely to be affected than arbitrary clauses. Notice that from the 201 possible child nodes at level 1 only 6% is not empty and not solved by the failed literals tests. It can be concluded that the branching strategy aims at affecting non-reverse Horn clauses, and hence increasing the (non-renamed) reverse Horn fraction in the child nodes compared to their parent node. The greedy heuristic can only increase the reverse Horn in a node by applying a sub-optimal renaming procedure.

3.9 Conclusions

We presented evidence for the fact that the k -level reverse Horn fraction may play a similar (albeit weaker) role as the density parameter in separating satisfiable random $\{3\}$ -SAT instances from unsatisfiable ones. If it is impossible to discriminate using the density parameter the k -level reverse Horn fraction seems suitable. From a statistical point of view it is favorable to combine weak separators. The k -level reverse Horn fraction can be combined with other different weak separators as, for instance, discussed in [10] and [145]. It is remarkable that the k -level reverse Horn fraction indeed has weakly separating properties because random $\{3\}$ -SAT instances start with having expected reverse Horn fraction equal to 0.5, and hence seem rather insensitive with respect to this parameter. It turned out that the k -level reverse Horn fraction suits this purpose better than a similarly obtained sub-optimal MAXSAT-fraction. Performances of various state-of-the-art solvers (complete as well as incomplete ones) seem to be correlated to the k -level reverse Horn fraction as well, although any link to or use of reverse Horn clauses in these solvers is absent. Recently, some theoretical back-up for the correlations observed come from [111] and [116]. In the first paper, a relation between Horn fraction and the size of "backdoor" sets is established. The size of these sets is correlated to computational effort of DLL based solving techniques. The second paper shows that larger Horn fractions yield lower complexity worst case bounds.

Chapter 4

Sums of Squares based approximation algorithms for MAX-SAT

4.1 Introduction

Hilbert's Positivstellensatz states that a non-negative polynomial in $\mathbb{R}[x_1, \dots, x_n]$ is a SOS in case $n = 1$, or has degree 2, or $n = 2$ and the degree is 4. Despite these restrictive constraints, explicit counter examples for the non-covered cases are rare, although Blekherman [15] proved that there must be many of them. On the other side Parrilo [113] claims that for purposes of optimization, the replacement of the fact that a polynomial is non-negative by the fact that it is a SOS gives very good results in practice. This claim could imply that we can develop an upper bound algorithm for MAX-SAT using the SOS approach, which gives tighter bounds than the existing ones.

Before entering the specific MAX-SAT context we first explain the SOS formalism: suppose a given polynomial $p(x_1, \dots, x_n) \in \mathbb{R}[x_1, \dots, x_n]$ has to be minimized over \mathbb{R}^n . This minimization can be written as the program

$$\begin{aligned} & \max \alpha \\ & \text{s.t. } p(x_1, \dots, x_n) - \alpha \geq 0 \text{ on } \mathbb{R}^n \\ & \alpha \in \mathbb{R} \end{aligned} \tag{4.1.1}$$

This chapter is based on a paper published in the proceedings of the SAT2005 conference [144] and a paper submitted to Discrete Applied Mathematics [143]

Clearly, the program

$$\begin{aligned} & \max \alpha && (4.1.2) \\ & \text{s.t. } p(x_1, \dots, x_n) - \alpha \text{ is a SOS} \\ & \alpha \in \mathbb{R} \end{aligned}$$

would result in a lower bound for program (4.1.1).

A benefit of the approach above is the possibility of using the theory of ‘Newton polytopes’. The exponent of a monomial $x_1^{a_1} \dots x_n^{a_n}$ is identified with a lattice point $\bar{a} = (a_1, \dots, a_n)$. The Newton polytope associated with a polynomial is the convex hull of all those lattice points associated with monomials appearing in the polynomial involved. Monomials useful for finding a SOS decomposition are those with an exponent \bar{a} for which $2\bar{a}$ is in the Newton polytope. Thus adding more monomials would not enlarge the chance of finding a SOS decomposition. This means that for purposes of global optimization of a polynomial over \mathbb{R}^n we have the advantage to know which monomials are possibly involved in a SOS decomposition (if existing) while we face the disadvantage that non-negative polynomials need not be decomposable as SOS.

Involving the Boolean constraints of the form $x_1^2 - 1 = 0, \dots, x_n^2 - 1 = 0$ the situation turns. Each polynomial that is non-negative on $\{-1, 1\}^n$ can be written as a SOS modulo the ideal $I_{\mathcal{B}}$ generated by the polynomials $x_1^2 - 1, \dots, x_n^2 - 1$. This result is a special case of a theorem by Putinar [120]. Note that we use $\{-1, 1\}$ -values for Boolean variables instead of the more commonly used $\{0, 1\}$ -values, which is much more attractive when algebraic formalisms are involved. However, in this case the ‘Newton polytope property’ is not valid, because higher degree monomials may cancel ones with lower degree, when performing calculations modulo $I_{\mathcal{B}}$. Hence, we have to consider possibly an exponential set of monomials in the SOS decomposition. To see this consider a polynomial $p(x_1, \dots, x_n)$, which is non-negative on $\{-1, 1\}^n$. The expression

$$SP(x_1, \dots, x_n) = \sum_{\sigma \in \{-1, 1\}^n} \frac{p(\sigma)}{2^n} (1 + \sigma_1 x_1) \dots (1 + \sigma_n x_n) \quad (4.1.3)$$

is easily seen to give the same outputs on $\{-1, 1\}^n$ as $p(x_1, \dots, x_n)$. Each $\frac{1 + \sigma_j x_j}{2}$ is a square modulo $I_{\mathcal{B}}$ because

$$\left(\frac{1 + \sigma_j x_j}{2} \right)^2 \equiv \frac{1 + \sigma_j x_j}{2} \text{ modulo } I_{\mathcal{B}} \quad (4.1.4)$$

Hence, $SP(x_1, \dots, x_n)$ is seen to be a SOS modulo $I_{\mathcal{B}}$. At the same time it becomes evident that we might need an exponentially large basis of monomials in realizing this decomposition. We see that if we want to optimize a polynomial over $\{-1, 1\}^n$ we have the advantage to know that a basis of monomials exists which will give an exact answer, while we are facing the disadvantage that this basis could be unacceptably large.

We now come to the point of explaining the SOS approach formally. Let $M^T = (M_1, \dots, M_k)$ be a row vector of monomials in variables x_1, \dots, x_n , and $p(x_1, \dots, x_n)$ a given polynomial in $\mathbb{R}[x_1, \dots, x_n]$. The equation $M^T L^T L M = p$ involving any matrix L of appropriate size would give an explicit decomposition of p as a SOS over the monomials used. Conversely, any SOS decomposition of p can be written in this way. This means that the Semidefinite Program

$$\begin{aligned} M^T S M &= p \\ S &\text{ positive semidefinite } (S \succeq 0) \end{aligned} \tag{4.1.5}$$

gives a polynomial time decision method for the question whether p can be written as a SOS using M_1, \dots, M_k as a basis of monomials (up to prescribed precision: the method uses real numbers represented with a certain precision). The constraint $M^T S M = p$ in fact results in a set of linear constraints in the entries of the matrix S . This is illustrated in example 1. To find a lower bound on the minimum of $p(x_1, \dots, x_n)$ we solve the program

$$\begin{aligned} \max \quad & \alpha \\ \text{s.t.} \quad & p - \alpha = M^T S M \\ & \alpha \in \mathbb{R}, S \succeq 0 \end{aligned} \tag{4.1.6}$$

Note that a $S \succeq 0$ has a Cholesky decomposition $S = L^T L$. If we consider the Boolean side constraints we have a similar program. In this case however the equation $M^T S M = p$ needs to be satisfied only modulo $I_{\mathcal{B}}$. Also this constraint results in a set of linear constraints in the entries of the matrix S , but different from the ones above. This is caused by the above mentioned cancelation effects.

Now we come to discuss the above in a MAX-SAT related context (for a general survey on the relations between semidefinite programming and satisfiability we refer to Anjos [6]). First we shall associate polynomials to CNF formulas: with a literal X_i we associate the polynomial $\frac{1}{2}(1 - x_i)$, and with $\neg X_j$ we associate $\frac{1}{2}(1 + x_j)$. With a clause we associate the products of the polynomials associated with its literals. Note that for a given assignment $\sigma \in \{-1, 1\}^n$ the polynomial associated with each clause outputs a zero or a one, depending of the fact whether σ satisfies the clause or not. With a CNF formula ϕ we associate two polynomials F_ϕ and $F_\phi^{\mathcal{B}}$. F_ϕ is the sum of squares of the polynomials associated with the clauses from ϕ . $F_\phi^{\mathcal{B}}$ is just the sum of those polynomials. Clearly, F_ϕ is non-negative on \mathbb{R}^n , and $F_\phi^{\mathcal{B}}$ is non-negative on $\{-1, 1\}^n$. $F_\phi(\sigma)$ and $F_\phi^{\mathcal{B}}(\sigma)$ give the number of clauses violated by assignment σ . The minima of F_ϕ and $F_\phi^{\mathcal{B}}$ yield upper bounds for the MAX-SAT solution.

The following two examples illustrate the construction of F_ϕ and $F_\phi^{\mathcal{B}}$ and the corresponding SDPs.

Example 1 Let ϕ be the CNF formula with the following three clauses

$$X \vee Y, \quad X \vee \neg Y, \quad \neg X$$

The polynomial F_ϕ is in this case

$$\begin{aligned} F_\phi(x, y) &= \left(\frac{1}{2}(1-x)\frac{1}{2}(1-y)\right)^2 + \left(\frac{1}{2}(1-x)\frac{1}{2}(1+y)\right)^2 + \left(\frac{1}{2}(1+x)\right)^2 \\ &= \frac{3}{8} + \frac{1}{4}x + \frac{3}{8}x^2 + \frac{1}{8}y^2 + \frac{1}{4}xy^2 + \frac{1}{8}x^2y^2 \end{aligned}$$

In order to attempt to find the maximal α such that $F_\phi - \alpha$ can be rewritten as a SOS it suffices to work with the monomial basis $M^T = (1, x, y, xy)$. The program we need to solve is

$$\begin{aligned} \max \alpha & \\ \text{s.t. } F_\phi - \alpha &= M^T S M \\ \alpha \in \mathbb{R}, S &\succeq 0 \end{aligned} \tag{4.1.7}$$

Let s_{ij} be the entry in S on row i and column j . When substituting F_ϕ , M and S program (4.1.7) becomes

$$\begin{aligned} \max \alpha & \\ \text{s.t. } \frac{3}{8} + \frac{1}{4}x + \frac{3}{8}x^2 + \frac{1}{8}y^2 + \frac{1}{4}xy^2 + \frac{1}{8}x^2y^2 - \alpha & \\ = s_{11} + (s_{12} + s_{21})x + (s_{13} + s_{31})y + s_{22}x^2 + s_{33}y^2 + s_{44}x^2y^2 & \\ + (s_{24} + s_{42})x^2y + (s_{43} + s_{34})xy^2 + (s_{14} + s_{41} + s_{23} + s_{32})xy & \\ \alpha \in \mathbb{R}, S &\succeq 0 \end{aligned} \tag{4.1.8}$$

The linear equalities for the s_{ij} are obtained by comparing the coefficients of the monomials on both sides of the equation. For example, if we consider the monomial xy^2 we have the equality

$$s_{43} + s_{34} = \frac{1}{4},$$

and for the monomial xy we have

$$s_{14} + s_{41} + s_{23} + s_{32} = 0$$

SDP (4.1.8) gives an output $\alpha = \frac{1}{3}$, from which we may conclude that $2\frac{2}{3}$ is an upper bound for the MAX-SAT solution of ϕ . Notice that $F_\phi = \frac{1}{3} + \frac{3}{8}\left(x + \frac{1}{3}\right)^2 + \frac{1}{8}(xy - y)^2$, i.e. $F_\phi - \frac{1}{3}$ is a SOS. For this ϕ , $F_\phi^{\mathcal{B}} = \frac{1}{2}(1-x)\frac{1}{2}(1-y) + \frac{1}{2}(1-x)\frac{1}{2}(1+y) + \frac{1}{2}(1+x) = 1$. Clearly, $F_\phi^{\mathcal{B}} = 1$ means that any assignment will exactly violate one clause.

Example 2 Let ϕ be the following CNF formula

$$X \vee Y \vee Z, \quad X \vee Y \vee \neg Z, \quad \neg Y \vee \neg T, \quad \neg X, \quad T$$

$$F_\phi^{\mathcal{B}}(x, y, z, t) = \frac{3}{2} + \frac{1}{4}x - \frac{1}{4}t + \frac{1}{4}xy + \frac{1}{4}yt \tag{4.1.9}$$

The Semidefinite Program (SDP)

$$\begin{aligned} & \max \alpha & (4.1.10) \\ & \text{s.t. } F_\phi^{\mathcal{B}} - \alpha \equiv (1, x, y, t)S(1, x, y, t)^T \text{ modulo } I_{\mathcal{B}} \\ & \alpha \in \mathbb{R}, S \succeq 0 \end{aligned}$$

can be rewritten using $x^2 \equiv y^2 \equiv z^2 \equiv t^2 \equiv 1 \text{ modulo } I_{\mathcal{B}}$ as

$$\begin{aligned} & \max \alpha & (4.1.11) \\ & \text{s.t. } \frac{3}{2} + \frac{1}{4}x - \frac{1}{4}t + \frac{1}{4}xy + \frac{1}{4}yt - \alpha \equiv \\ & s_{11} + s_{22} + s_{33} + s_{44} + (s_{12} + s_{21})x + (s_{13} + s_{31})y + (s_{14} + s_{41})t + \\ & (s_{23} + s_{32})xy + (s_{24} + s_{42})tx + (s_{43} + s_{34})yt \text{ modulo } I_{\mathcal{B}} \\ & \alpha \in \mathbb{R}, S \succeq 0 \end{aligned}$$

Program (4.1.11) gives output $\alpha = 0.793$, from which we may conclude that 4.207 is an upper bound for the MAX-SAT solution of ϕ . The SDP

$$\begin{aligned} & \max \alpha & (4.1.12) \\ & \text{s.t. } F_\phi^{\mathcal{B}} - \alpha \equiv (1, x, y, t, xy, xt, yt)S(1, x, y, t, xy, xt, yt)^T \text{ modulo } I_{\mathcal{B}} \\ & \alpha \in \mathbb{R}, S \succeq 0 \end{aligned}$$

gives output $\alpha = 1$. Note that the second SDP gives a tighter upper bound, because the basis contains more monomials.

Below we formulate some properties of the polynomials F_ϕ and $F_\phi^{\mathcal{B}}$. Let m be the number of clauses, and n the number of variables in CNF-formula ϕ .

Theorem 4.1.1. (1) For any assignment $\sigma \in \{-1, 1\}^n$, $F_\phi(\sigma) = F_\phi^{\mathcal{B}}(\sigma)$. Both give the number of clauses violated by σ .

(2) $\min_{\sigma \in \{-1, 1\}^n} F_\phi(\sigma)$ and $\min_{\sigma \in \{-1, 1\}^n} F_\phi^{\mathcal{B}}(\sigma)$ give rise to an exact MAX-SAT solution of ϕ : respectively $m - \min_{\sigma \in \{-1, 1\}^n} F_\phi(\sigma)$ and $m - \min_{\sigma \in \{-1, 1\}^n} F_\phi^{\mathcal{B}}(\sigma)$.

(3) $F_\phi^{\mathcal{B}} \equiv F_\phi \text{ modulo } I_{\mathcal{B}}$.

(4) F_ϕ attains its minimum over \mathbb{R}^n somewhere in the hypercube $[-1, 1]^n$ (a compact set), while it can be zero only in a partial satisfying assignment.

(5) ϕ is unsatisfiable if and only if there exists an $\epsilon > 0$ such that $F_\phi - \epsilon \geq 0$ on \mathbb{R}^n .

(6) If there exists an $\epsilon > 0$ such that $F_\phi - \epsilon$ is a SOS, then ϕ is unsatisfiable.

(7) If there exists a monomial basis M and an $\epsilon > 0$ such that $F_\phi^{\mathcal{B}} - \epsilon$ is a SOS based on M , modulo $I_{\mathcal{B}}$, then ϕ is unsatisfiable.

(8) Let M be a monomial basis, then

$$\begin{aligned} m - \max \alpha & & (4.1.13) \\ \text{s.t. } F_\phi - \alpha & \text{ is a SOS} \\ \alpha & \in \mathbb{R} \end{aligned}$$

and

$$\begin{aligned} m - \max \alpha & & (4.1.14) \\ \text{s.t. } F_\phi^{\mathcal{B}} - \alpha & \equiv M^T S M \text{ modulo } I_{\mathcal{B}} \\ \alpha & \in \mathbb{R}, S \succeq 0 \end{aligned}$$

give upper bounds for the MAX-SAT solution of ϕ .

Proof. Except for part 1.4 the reasonings behind the other parts are already discussed before or they are direct consequences of earlier statements. Here we prove Theorem 1.4: suppose F_ϕ takes its minimum in x , and assume $x_1 = 1 + \delta$ for some $\delta > 0$. This gives rise to contributions $(\frac{1}{2}(1 + (1 + \delta)))^2$ and $(\frac{1}{2}(1 - (1 + \delta)))^2$. Both contributions are smaller with $\delta = 0$ than with $\delta > 0$. The same argument can be applied for $x_1 = -1 - \delta$. Thus $x \in [-1, 1]^n$. Furthermore, $F_\phi = 0$ only if in each polynomial associated to a clause at least one of the factors equals zero because F_ϕ is a sum of squares, and hence non-negative. This can only be realized if in each polynomial associated to a clause at least one of the variables takes value 1 or -1 resulting in a partial satisfying assignment for ϕ . \square

Program (4.1.14) is the basis for the search for MAX-SAT upper bounds in this paper. Theorems 1.5 and 1.8, program (4.1.13), could serve as a starting point for the search for counterexamples for the non-covered cases of Hilbert's Positivstellensatz. Clearly, a 2-SAT formula ϕ gives a polynomial F_ϕ with degree 4, and the SDP (4.1.13) must have $\alpha = 0$ for a satisfiable formula (F_ϕ is a SOS itself by construction). For an unsatisfiable formula ϕ , the optimal α might be zero, in which case $F_\phi - \epsilon$, with ϵ sufficiently small, is a non-negative polynomial, but not a SOS. The optimal α might be positive, in which case ϕ does not give a counterexample but gives a proof of the unsatisfiability of the instance. We will report on some experiments and prove a theorem relating complexity issues to the existence of counterexamples of Hilbert's Positivstellensatz of a specific form in Section 4.8.

We close this section with a classification based on Theorem 1. Let ϕ be a CNF-formula. As we have seen before we have

Theorem 4.1.2. *ϕ is unsatisfiable if and only if $F_\phi^{\mathcal{B}} - \epsilon$ is a SOS modulo $I_{\mathcal{B}}$ for some $\epsilon > 0$.*

Let I_ϕ be the ideal generated by $F_\phi^{\mathcal{B}}$ and $I_{\mathcal{B}}$. We can formulate the rather elegant theorem whose computational implications are not transparent yet.

Theorem 4.1.3. *ϕ is unsatisfiable if and only if -1 is a sum of squares in the ring $\mathbb{R}[x_1, \dots, x_n]$ modulo I_ϕ .*

At the end of this section we will define the notation of five monomial bases that are used throughout the remainder of this paper. $1, x_1, \dots, x_n$ are contained in any of these bases. A product $x_i x_j$ occurs in the polynomial related to the CNF-formula (at least before adding terms with the same monomial) if X_i and X_j occur in a same clause. This makes these monomials probably good choices to include in the monomial basis.

Definition 4.1.4. • M_{GW} is the monomial basis containing $1, x_1, \dots, x_n$ (applicable for MAX-2-SAT).

- M_p is the monomial basis containing $1, x_1, \dots, x_n$ and all monomials $x_i x_j$ for variables X_i and X_j appearing in a same clause (applicable for MAX-2-SAT and MAX-3-SAT).
- M_{ap} is the monomial basis containing $1, x_1, \dots, x_n$ and monomials $x_i x_j$ for each pair of variables X_i and X_j (applicable for MAX-2-SAT and MAX-3-SAT).
- Monomial basis M_t contains $1, x_1, \dots, x_n$ and the monomials $x_i x_j x_k$ such that X_i, X_j and X_k occur in a same clause (applicable for MAX-3-SAT).
- Monomial basis M_{pt} contains $1, x_1, \dots, x_n$, all monomials $x_i x_j$ for variables X_i and X_j appearing in a same clause and all monomials $x_i x_j x_k$ such that X_i, X_j and X_k occur in a same clause (applicable for MAX-3-SAT).

The corresponding SDPs with computation modulo I_B are called SOS_{GW} , SOS_p , SOS_{ap} , SOS_t and SOS_{pt} respectively. Note that when M_t is selected as monomial basis, monomials of degree two are obtained as products of variables and monomials of degree three modulo I_B . Secondly, note that M_{GW} can only be used as monomial basis if all clauses have length smaller than or equal to two. If a CNF-formula ϕ contains a clause of length three, F_ϕ^B probably contains a monomial of the form $x_i x_j x_k$, which does not occur in $M_{GW}^T S M_{GW}$.

In section 4.2 we prove that SOS_{GW} gives upper bounds equal to the ones obtained by the approach by Goemans and Williamson. Furthermore, we prove that adding monomials to the monomial basis has the same, and possibly stronger, effect as adding related valid inequalities to the SDP of Goemans and Williamson. In section 4.2.3 we prove that SOS_{ap} always finds an upper bound equal to the optimum for a class of problems having worst case known performance for the approach of Feige and Goemans. Experimental results for the quality of the different upper bounding techniques for randomly generated MAX-2-SAT instances are presented in section 4.2.4. Section 4.3 gives the computational complexities of the different approaches given a particular SDP algorithm (Sedumi [134]). In section 4.4 we present experimental results for random MAX-3-SAT and a proof showing that the upper bounds obtained by SOS_{pt} are at least as tight as the one obtained by the method by Karloff and Zwick.

Section 4.5 experimentally compares SOS_t and SOS_{pt} with a relaxation by Anjos for proving unsatisfiability of 3-SAT instances. The new SOS-based rounding procedure used for obtaining lower bounds is described and experimentally investigated in section 4.6. In section 4.7 we give experimental evidence that in many cases a considerable fraction of the constraints in the SOS SDPs only fixes variables. Section 4.8 deals with the interrelationship between the complexity of solving SDPs and counterexamples to uncovered cases of Hilbert's Positivstellensatz stemming from unsatisfiable CNF-formulas.

4.2 SDP-based upper bounds for MAX-2-SAT

Although the SOS approach provides upper bounds for general MAX-SAT instances, we restrict ourselves in this section to MAX-2-SAT. The reason is that we want to present a comparison with the results for the famous methods of Goemans and Williamson [60] and Feige and Goemans [53]. In this section, we consider SOS_{GW} .

Semidefinite Programming formulations come in pairs: the so-called primal and dual formulations, see for example [42] and [39]. Here we present a generic formulation of a primal semidefinite problem P , and its dual program D . In the context of this paper, D and P give the same optimal value.

Consider the program P

$$\min \text{Tr}(CX) \tag{4.2.1}$$

$$\text{s.t. } \text{diag}(X) = e \tag{4.2.2}$$

$$\text{Tr}(A_j X) \geq 1, \quad j = 1, \dots, k \tag{4.2.3}$$

$$X \succeq 0$$

In the above formulation, C and X are symmetric square matrices of size, say $p \times p$. Tr means the trace of the matrix. This equals the sum of the entries on the diagonal, i.e.

$$\text{Tr}(CX) = \sum_{i=1}^p \sum_{j=1}^p c_{ij} x_{ij}$$

$\text{diag}(X) = e$ means that the entries on the diagonal of matrix X are all ones. The A_j s are square symmetric matrices.

The program P has the following dual program D

$$\max \sum_{i=1}^p \gamma_i + \sum_{j=1}^k y_j \tag{4.2.4}$$

$$\text{s.t. } \text{Diag}(\gamma) + \sum_{j=1}^k y_j A_j + U = C$$

$$U \succeq 0, y_j \geq 0,$$

in which the y_j s are the dual variables corresponding to constraints (4.2.3), and the γ_i s the dual variables corresponding to constraint (4.2.2). U is a symmetric square matrix, and $\text{Diag}(\gamma)$ is the square matrix with the γ_i s on the diagonal and all off-diagonal entries equal to zero.

4.2.1 Comparison of SOS_{GW} and Goemans-Williamson approach

The original Goemans-Williamson approach for obtaining an upper bound for the MAX-2-SAT problem starts with $F_\phi^{\mathcal{B}}$ too. The problem

$$\begin{aligned} \min F_\phi^{\mathcal{B}}(x) \\ x \in \{-1, 1\}^n \end{aligned} \quad (4.2.5)$$

is relaxed by relaxing the Boolean arguments x_i . With each x_i , a vector $v_i \in \mathbb{R}^{n+1}$ is associated, with norm 1, and products $x_i x_j$ are interpreted as inproducts $v_i \bullet v_j$. They make $F_\phi^{\mathcal{B}}$ homogenous by adding a dummy vector v_0 in order to make the linear terms in $F_\phi^{\mathcal{B}}$ quadratic as well. For example, $3x_i$ is replaced by $3(v_i \bullet v_0)$. Let $\hat{F}_\phi^{\mathcal{B}}$ be the polynomial constructed from $F_\phi^{\mathcal{B}}$ in this way. The problem Goemans and Williamson solve is

$$\begin{aligned} \min \hat{F}_\phi^{\mathcal{B}}(v_0, v_1, \dots, v_n) \\ \text{s.t. } \|v_i\| = 1, v_i \in \mathbb{R}^{n+1} \end{aligned} \quad (4.2.6)$$

To transform (4.2.6) to a semidefinite program, $v_i \bullet v_j$ is replaced by t_{ij} . Let b_{ij} be the coefficient of $(M_{GW})_i (M_{GW})_j$ in the polynomial $F_\phi^{\mathcal{B}}$. Let $f_{ij} = f_{ji} = \frac{1}{2} b_{ij}$ and $f_{ii} = 0$ for each i . Let $M(F)$ be the symmetric matrix with entries f_{ij} and T be a symmetric matrix of the same size. Furthermore, let c_0 be the constant term in $F_\phi^{\mathcal{B}}$. To be precise, c_0 equals $\frac{1}{2}$ times the number of 1-literal clauses plus $\frac{1}{4}$ times the number of 2-literal clauses in ϕ .

Consequently, (4.2.6) is equivalent to the following semidefinite program

$$\begin{aligned} c_0 + \min \sum_{i=1}^{n+1} \sum_{j=1}^{n+1} f_{ij} t_{ij} \\ \text{s.t. } t_{ii} = 1, T \succeq 0 \end{aligned} \quad (4.2.7)$$

or in matrix notation,

$$\begin{aligned} c_0 + \min \text{Tr}(M(F)T) \\ \text{s.t. } \text{diag}(T) = e, T \succeq 0 \end{aligned} \quad (4.2.8)$$

While Goemans and Williamson relax the input arguments of $F_\phi^{\mathcal{B}}$, the SOS-approach is a relaxation by replacing non-negativity by being a SOS. The next theorem proves that SOS_{GW} gives the same upper bound for MAX-2-SAT as program (4.2.8).

Theorem 4.2.1. *SOS_{GW} gives the same upper bound as the algorithm of Goemans and Williamson.*

Proof. In the Goemans-Williamson SDP (4.2.8) we only have the constraints of type (4.2.2), and not of type (4.2.3). This implies that we have to deal only with the γ_j -variables. The size of the variable matrix T is $n + 1$.

The dual problem of the Goemans-Williamson-semidefinite program (4.2.8) is

$$\begin{aligned} c_0 + \max \sum_{i=1}^{n+1} \gamma_i & \quad (4.2.9) \\ \text{s.t. } \text{Diag}(\gamma) + U = M(F) & \\ U \succeq 0, \gamma_i \text{ free} & \end{aligned}$$

We start the proof with the program

$$\begin{aligned} \max \alpha & \quad (4.2.10) \\ \text{s.t. } F_\phi^{\mathcal{B}} - \alpha \equiv M^T S M \text{ modulo } I_{\mathcal{B}} & \\ S \succeq 0, \alpha \in \mathbb{R} & \end{aligned}$$

with monomial basis $M = M_{GW} = ((M_{GW})_1, \dots, (M_{GW})_{n+1}) = (1, x_1, \dots, x_n)$ and prove that it is equal to (4.2.9). Let s_{ij} be the (i, j) -th element of matrix S .

We can reformulate program (4.2.10) as

$$\begin{aligned} \max \alpha & \quad (4.2.11) \\ \text{s.t. } \sum_{i=1}^{n+1} \sum_{j=1}^{n+1} s_{ij} (M_{GW})_i (M_{GW})_j \equiv F_\phi^{\mathcal{B}} - \alpha \text{ modulo } I_{\mathcal{B}} & \\ S \succeq 0, \alpha \in \mathbb{R} & \end{aligned}$$

Consider the constraint in program (4.2.11) for the coefficient of the constant. On the left hand side we have $\sum_{i=1}^{n+1} s_{ii}$ because $((M_{GW})_i)^2 \equiv 1 \text{ modulo } I_{\mathcal{B}}$. On the right hand side we have $c_0 - \alpha$. This results in the equality

$$\alpha = c_0 - \sum_{i=1}^{n+1} s_{ii} \quad (4.2.12)$$

In the matrix formulation (4.2.13), on both left and right hand sides we have a matrix with on position (i, j) , $i \neq j$, the coefficient of $(M_{GW})_i (M_{GW})_j$ using symmetry. Substituting (4.2.12) and using matrix notation we can reformulate (4.2.11) as

$$\begin{aligned} c_0 + \max \sum_{i=1}^{n+1} -s_{ii} & \quad (4.2.13) \\ \text{s.t. } S - \text{diag}(S) = M(F) & \\ S \succeq 0 & \end{aligned}$$

Identifying γ_i with matrix entries $-s_{ii}$ and U with S , it is immediate that (4.2.13) is equivalent to (4.2.9).

Hence, we proved that (4.2.10) with monomial basis M_{GW} equals (4.2.9). It can be concluded that the Goemans-Williamson SDP and SOS_{GW} are dual problems providing the same upper bounds for MAX-2-SAT instances. \square

Still, there is something more to say about these two different approaches. Program (4.2.7) has $\frac{1}{2}(n+1)(n+2)$ variables t_{ij} (not $(n+1)^2$ because T is symmetric).

In SOS_{GW} (4.1.14), each product of two different monomials yields a unique monomial. This means that each equality is of the form

$$s_{ij} + s_{ji} = c$$

for some constant c . For each pair $(i, j), i \neq j$, there is such an equality. Due to symmetry this implies that in fact only the diagonal elements are essentially variable, because all off-diagonal elements are fixed. This means that the actual dimension of the SOS-program with monomial basis M_{GW} is linear in the number of variables, while in the Goemans-Williamson formulation (4.2.7) the dimension grows quadratically.

In the experiments we tried several semidefinite programming solvers like Sedumi [134], DSDP [13], CSDP [19] and SDPA [58], but none of them could fully benefit from this fact. However, we found that CSDP performed best on SDPs of the form (4.1.14). In section 4.7 we investigate how many constraints in the SOS approach are in fact 'unit' constraints.

4.2.2 Adding valid inequalities vs adding monomials

Feige and Goemans [53] propose to add so-called valid inequalities to (4.2.7) in order to improve the quality of the relaxation. A valid inequality is an inequality that is satisfied by any optimal solution of the original (unrelaxed) problem but may be violated by the optimal solution of the relaxation. Valid inequalities improve the quality of the relaxation because they exclude a part of its feasible region that cannot contain the optimal solution of the original problem. Triangle inequalities are among the most frequently used valid inequalities. Two types of these 'triangle inequalities' are considered. The first is the inequality

$$1 + x_i + x_j + x_i x_j \geq 0 \tag{4.2.14}$$

Note that in (4.2.7) t_{ij} has replaced $x_i x_j$, and $t_{i,n+1}$ replaces x_i from F_ϕ^B . In fact, they consider the inequality $1 + t_{i,n+1} + t_{j,n+1} + t_{ij} \geq 0$, which is added to (4.2.7). All these inequalities can be added, also the ones obtained by replacing x_i and/or x_j by $-x_i$ or $-x_j$. Another possibility is to add only those inequalities where X_i and X_j appear together in the same clause. In this section, we examine how these valid inequalities compare with SOS_p . It can be shown that $1 + x_i + x_j + x_i x_j$ cannot be recognized as

a SOS based on $M = (1, x_i, x_j)$. However, if we add $x_i x_j$ to the monomial basis we have

$$1 + x_i + x_j + x_i x_j \equiv \frac{1}{4} (1 + x_i + x_j + x_i x_j)^2 \text{ modulo } I_B$$

A similar argument can be given for the three inequalities with x_i and/or x_j replaced by $-x_i$ and/or $-x_j$. Hence, the effect of adding the valid inequality (4.2.14) and the three similar inequalities is captured in the SOS approach by adding the monomial $x_i x_j$ to the basis. Below we prove a theorem from which follows that adding the monomial $x_i x_j$ results in upper bounds that are at least as tight as the upper bound of the Goemans-Williamson program (4.2.7) together with the four triangle inequalities of the form (4.2.14). The experiments in section 4.2.4 support this fact.

Feige and Goemans [53] further showed that adding for each triple of variables X_i, X_j and X_k to (4.2.7) the valid inequalities

$$\begin{aligned} 1 + t_{ij} + t_{ik} + t_{jk} &\geq 0, & 1 - t_{ij} + t_{ik} - t_{jk} &\geq 0 \\ 1 + t_{ij} - t_{ik} - t_{jk} &\geq 0, & 1 - t_{ij} - t_{ik} + t_{jk} &\geq 0 \end{aligned} \quad (4.2.15)$$

improves the tightness of the relaxation. Note that

$$1 + x_i x_j + x_i x_k + x_j x_k \equiv \frac{1}{4} (1 + x_i x_j + x_i x_k + x_j x_k)^2 \text{ modulo } I_B$$

Hence, the effect of adding the four inequalities (4.2.15) is captured by adding $x_i x_j, x_i x_k$ and $x_j x_k$ to the monomial basis. Also in this case, the effect of adding the monomials to the monomial basis results in upper bounds at least as tight compared to adding valid inequalities to the Goemans-Williamson SDP as shown in Theorem 4.2.2.

Finally, note that adding all inequalities of the form (4.2.15) amounts to adding $\mathcal{O}(n^3)$ inequalities, while in the SOS approach $\mathcal{O}(n^2)$ monomials of degree 2 need to be added. For the moment, it is too early to decide whether existing SDP-solvers are suitable, or can be modified, to turn this feature into a computational benefit as well.

Theorem 4.2.2. *Adding monomials $x_i x_j, x_i x_k$ and $x_j x_k$ to the monomial basis in the SOS approach gives an upper bound at least as tight as the upper bound obtained by adding triangle inequalities of the type (4.2.15) to the Goemans-Williamson SDP (4.2.7).*

Proof. Without loss of generality we consider the triangle inequality

$$1 + x_1 x_2 - x_1 x_3 - x_2 x_3 \geq 0 \quad (4.2.16)$$

In the notation of (4.2.7) this equation is $1 + t_{12} - t_{13} - t_{23} \geq 0$. In matrix notation this inequality is $Tr(AT) \geq 1$ with

$$A = \begin{pmatrix} 1 & 1 & -1 \\ 1 & 1 & -1 \\ -1 & -1 & 1 \end{pmatrix}$$

We consider the program

$$\begin{aligned}
& \min \operatorname{Tr}(M(F)T) & (4.2.17) \\
& \text{s.t. } \operatorname{diag}(T) = e \\
& \operatorname{Tr}(AT) \geq 1 \\
& T \succeq 0
\end{aligned}$$

Assume that F is an homogenous polynomial of degree 2 in three variables x_1, x_2, x_3 only. This does not harm the general validity of this proof but makes the key steps more transparent. $M(F)$ is the coefficient matrix associated with the polynomial F . Let $F(x_1, x_2, x_3) = 2ax_1x_2 + 2bx_1x_3 + 2cx_2x_3$. Then,

$$M(F) = \begin{pmatrix} 0 & a & b \\ a & 0 & c \\ b & c & 0 \end{pmatrix}$$

The dual program of (4.2.17) is the following

$$\begin{aligned}
& \max \gamma_1 + \gamma_2 + \gamma_3 + y & (4.2.18) \\
& \text{s.t. } yA + \operatorname{Diag}(\gamma) + U = M(F) \\
& U \succeq 0, y \geq 0
\end{aligned}$$

with $\operatorname{Diag}(\gamma)$ the 3×3 -matrix with on its diagonal $\gamma_1, \gamma_2, \gamma_3$.

Program (4.2.18) can be reformulated as

$$\begin{aligned}
& \max \gamma_1 + \gamma_2 + \gamma_3 + y & (4.2.19) \\
& \text{s.t. } M(F) - \operatorname{Diag}(\gamma) - yA \succeq 0 \\
& y \geq 0
\end{aligned}$$

Now suppose that $(\hat{\gamma}_1, \hat{\gamma}_2, \hat{\gamma}_3, \hat{y})$ is an optimal solution for (4.2.19). We will show that from this optimal solution a feasible solution for

$$\begin{aligned}
& \max \alpha & (4.2.20) \\
& \text{s.t. } MSM^T \equiv F - \alpha \pmod{I_B}
\end{aligned}$$

can be constructed with $M = (1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3)$. In fact, we will even show that the monomial basis $M_1 = (1, x_1x_2, x_1x_3, x_2x_3)$ is already sufficient in this respect.

Program (4.2.20) with monomial basis M_1 can be reformulated as

$$\begin{aligned}
& \max \left(- \sum_{i=1}^4 s_{ii} \right) & (4.2.21) \\
& s_{12} + s_{21} + s_{34} + s_{43} = 2a \\
& s_{13} + s_{31} + s_{24} + s_{42} = 2b \\
& s_{23} + s_{32} + s_{14} + s_{41} = 2c \\
& S \succeq 0
\end{aligned}$$

$1 + x_1x_2 - x_1x_3 - x_2x_3$ is a SOS modulo $I_{\mathcal{B}}$, because the following holds

$$1 + x_1x_2 - x_1x_3 - x_2x_3 = \frac{1}{4}M_1\Delta M_1^T$$

with Δ the positive semidefinite matrix

$$\Delta = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{pmatrix}$$

Let Z be the 4×4 matrix with the 3×3 matrix $M(F) - \text{Diag}(\hat{\gamma}) - \hat{y}A$ starting in the upper left corner and having zeros in fourth row and column. We can conclude that $Z + \frac{1}{2}\hat{y}\Delta \succeq 0$ because $Z \succeq 0$, $\Delta \succeq 0$ and $\hat{y} \geq 0$. The matrix $Z + \frac{1}{2}\hat{y}\Delta$

$$Z + \frac{1}{2}\hat{y}\Delta = \begin{pmatrix} -\hat{\gamma}_1 - \frac{1}{2}\hat{y} & a - \frac{1}{2}\hat{y} & b + \frac{1}{2}\hat{y} & -\frac{1}{2}\hat{y} \\ a - \frac{1}{2}\hat{y} & -\hat{\gamma}_2 - \frac{1}{2}\hat{y} & c + \frac{1}{2}\hat{y} & -\frac{1}{2}\hat{y} \\ b + \frac{1}{2}\hat{y} & c + \frac{1}{2}\hat{y} & -\hat{\gamma}_3 - \frac{1}{2}\hat{y} & -\frac{1}{2}\hat{y} \\ -\frac{1}{2}\hat{y} & -\frac{1}{2}\hat{y} & -\frac{1}{2}\hat{y} & \frac{1}{2}\hat{y} \end{pmatrix}$$

satisfies the constraints in (4.2.21) and $-\sum_{i=1}^4 s_{ii} = \hat{\gamma}_1 + \hat{\gamma}_2 + \hat{\gamma}_3 + \hat{y}$. \square

Because the optimal solution of Goemans-Williamson SDP with Feige-Goemans valid inequalities equals a feasible solution of SOS_p , it can be concluded that the SOS approach gives at least as tight upper bounds.

At this point, we proved in this section that the upper bounds obtained by SOS_{GW} and the ones obtained by the approach of Goemans and Williamson are equal. Furthermore, we proved that SOS_p gives upper bounds that are at least as tight as the ones obtained by the Feige-Goemans approach with four valid inequalities for each pair of variables occurring in a same clause. Furthermore, we showed that SOS_{ap} provides upper bounds at least as tight as the Feige-Goemans approach with all inequalities of the form (4.2.15) added.

4.2.3 SOS_{ap} on worst known case for Feige-Goemans

The CNF formula of ϕ_{FGn} in n variables is defined as

$$\begin{aligned} & x_1 \vee x_2, x_2 \vee x_3, x_3 \vee x_4, \dots, x_n \vee x_1 \\ & \neg x_1 \vee \neg x_2, \neg x_2 \vee \neg x_3, \neg x_3 \vee \neg x_4, \dots, \neg x_n \vee \neg x_1 \end{aligned} \tag{4.2.22}$$

Feige and Goemans [53] present ϕ_{FG5} as worst-known case example with respect to the performance guarantee of their approach.

Note that we can satisfy $2n-1$ of the clauses if n is odd by setting the odd-numbered variables to true and the even-numbered variables to false. It is not possible to satisfy all clauses for odd n . In this section, we show that the SOS_{ap} finds the optimal MAX-SAT solution of ϕ_{FGn} .

Theorem 4.2.3. *Let n be an odd number. The polynomial $F_{\phi_n}^{\mathcal{B}} - 1$ with*

$$F_{\phi_n}^{\mathcal{B}} = \frac{1}{2} (n + x_1x_2 + x_2x_3 + \cdots + x_{n-1}x_n + x_nx_1) \quad (4.2.23)$$

is a sum of squares if we choose as monomial basis M_{ap} .

Proof. As initial step we start with ϕ_{FG3} . The polynomial $F_{\phi_3}^{\mathcal{B}}$ is

$$F_{\phi_3}^{\mathcal{B}} = \frac{1}{2} (3 + x_1x_2 + x_2x_3 + x_3x_1) \quad (4.2.24)$$

Define $F_3(x) = 1 + x_1x_2 + x_2x_3 + x_3x_1$. $F_{\phi_3}^{\mathcal{B}} - 1$ is a sum of squares modulo $I_{\mathcal{B}}$, because

$$\frac{1}{2} \left(\frac{1}{2} F_3(x) \right)^2 \equiv \frac{1}{2} F_3(x) \text{ modulo } I_{\mathcal{B}}$$

We use this fact to prove by induction that $F_{\phi_n}^{\mathcal{B}} - 1$ is a sum of squares relative to the monomial basis considered. Assume that the polynomial $F_{\phi_{n-2}}^{\mathcal{B}} - 1$ related to $\phi_{FG(n-2)}$ is a sum of squares modulo $I_{\mathcal{B}}$.

The polynomial $F_{\phi_n}^{\mathcal{B}}$ equals

$$F_{\phi_n}^{\mathcal{B}} = F_{\phi_{n-2}}^{\mathcal{B}} + \frac{1}{2} (2 + x_{n-2}x_{n-1} + x_{n-1}x_n + x_nx_1 - x_{n-2}x_1) \quad (4.2.25)$$

We assumed that $F_{\phi_{n-2}}^{\mathcal{B}} - 1$ is a sum of squares. Let $T_1(x) = 1 - x_1x_{n-2} + x_{n-2}x_{n-1} + x_1x_{n-1}$ and $T_2(x) = 1 - x_1x_{n-1} + x_{n-1}x_n + x_nx_1$. Note that $F_{\phi_n}^{\mathcal{B}} = F_{\phi_{n-2}}^{\mathcal{B}} + \frac{1}{2}(T_1(x) + T_2(x))$ and for $i = 1$ and $i = 2$

$$\frac{1}{2} \left(\frac{1}{2} T_i(x) \right)^2 \equiv \frac{1}{2} T_i(x) \text{ modulo } I_{\mathcal{B}}$$

This proves that $F_{\phi_n}^{\mathcal{B}} - 1$ is also a sum of squares. □

From this theorem we can conclude that SOS_{ap} identifies $F_{\phi_n}^{\mathcal{B}} - 1$ as a sum of squares. Hence, the minimum of $F_{\phi_n}^{\mathcal{B}}$ is at least 1. We conclude that SOS_{ap} solves (4.2.22) to optimality.

4.2.4 Experimental results for random MAX-2-SAT

In this section we consider besides the Goemans-Williamson upper bound the next four variants of the Feige-Goemans method.

Variante FG_m : The valid inequalities added in this variant are only those coming directly from the clauses. For instance, if $X \vee \neg Y$ is a clause, we add the valid inequality $1 - x + y - xy \geq 0$.

Variante FG_{4p} : For each pair of variables X_i and X_j occurring in a same clause, the four inequalities of the type (4.2.14) are added.

Variante FG_{ap} : For *each* pair of variables, the four inequalities of the type (4.2.14) are added.

Variante FG_{pt} : All inequalities of variant FG_{ap} are added and additionally for each triple of variables the four inequalities of type (4.2.15) added.

We compare the upper bounds resulting from these variants with the upper bounds obtained from the semidefinite program (4.1.14) with monomial basis M_p . We call the corresponding upper bound SOS_p . SOS_{ap} is the variant with monomial basis $\{1, x_1, \dots, x_n\}$ extended with all $x_i x_j$ for each pair of variables X_i and X_j . We will present results for small-scale problems only in this section, because solving the SDPs of SOS_p takes a lot of time with the SDP-solvers currently available.

d	SOS_p	SOS_{ap}	GW	FG_m	FG_{4p}	FG_{ap}	FG_{pt}
1.0	1	1	0.933480	0.984195	0.993151	0.993151	1
1.5	1	1	0.953544	0.989779	0.993869	0.993869	1
2.0	0.99984	1	0.969460	0.994136	0.997043	0.997242	1
2.5	1	1	0.979549	0.996760	0.998317	0.998420	1
3.0	1	1	0.981904	0.996485	0.998044	0.998188	1
3.5	1	1	0.985519	0.997435	0.998904	0.998975	1
4.0	0.999995	1	0.987250	0.997538	0.998873	0.998930	0.999964
4.5	0.999973	0.999973	0.986327	0.997120	0.998692	0.998776	0.999936
5.0	0.999979	0.999979	0.987015	0.997779	0.998764	0.998841	0.999971

Table 4.1: 10 variables, random MAX-2-SAT, bound ratios

In initial experiments we used a set of 900 randomly generated instances with 10 variables and different clause-variable densities d . The (clause-variable) density is the number of clauses divided by the number of variables. For any of the densities 1.0, 1.5, 2.0, \dots , 5.0, 100 instances are considered. The *bound ratio* R is defined as the optimal MAX-SAT solution divided by the upper bound found. The bound ratio can be seen as a size independent measure for the quality of the upper bounds.

In Table 4.1 we give for each method the average R over the set of unsatisfiable instances out of the 100 generated instances for each density. The first column indicates the density, the second column the bound ratio for SOS_p , the third column gives the results for SOS_{ap} , the fourth gives the Goemans-Williamson bound ratio, the fifth gives the bound ratio of variant FG_m , the next the bound ratio of variant FG_{4p} , then the bound ratio of FG_{ap} and the last column gives the bound ratio of variant FG_{pt} .

From Table 4.1 we see that the upper bounds obtained by SOS_{ap} are at least as tight as the other ones. This is not only true on average but in fact for each individual instance involved in our experiments. For the selected set of instances, SOS_p turns out to be almost always, except for one instance, at least as good as the best Feige-Goemans variant FG_{pt} while this is not forced by Theorem 4.2.2. In these experiments with MAX-2-SAT instances with 10 variables, the SDPs of each variant are solved by Sedumi [134]. Table 4.2 gives the same type of results for instances with 25 variables but only for the methods that are most relevant and computationally not too expensive. For the instances with 25 variables, GW and FG_{ap} are solved by

Sedumi. The SDPs of SOS_p are solved by CSDP [19], because this solver is faster, more accurate, and uses less memory when solving the SDPs of SOS_p .

d	SOS_p	GW	FG_{ap}
1.5	0.999403	0.955241	0.995686
2.0	0.999607	0.968157	0.997279
2.5	0.999577	0.976133	0.997639
3.0	0.999906	0.979769	0.998238
3.5	0.999691	0.981444	0.997520
4.0	0.999908	0.982812	0.997890
4.5	0.999882	0.983297	0.997721
5.0	0.999989	0.984816	0.998196

Table 4.2: 25 variables, random MAX-2-SAT, bound ratios

The detailed results for the experiment with instances with 25 variables are given in Tables 4.3. In these tables, the first column gives the density, the second gives the optimal MAX-SAT solution. In the third column is indicated the number of instances for which the upper bound found by SOS_p equals the optimal MAX-SAT solution, in which a small numerical error of at most 10^{-5} is allowed. Columns four and five respectively give the average and minimal bound ratio for SOS_p . The last is an indication for the worst case result on the test set. Columns 6, 7 and 8 give similar results for FG_{ap} . The last column mentions the number of instances with the MAX-SAT solution equal to the number in the second column. From Tables 4.3 is clear that SOS_p finds an upper bound equal to the optimal MAX-SAT solution considerably more often than FG_{ap} .

4.3 Complexity of different approaches wrt short step semidefinite optimization algorithms

The complexity of short step semidefinite optimization algorithms (like for example Sedumi) is $\mathcal{O}((2V^2 + C)\sqrt{V})$ if V is the size of the semidefinite variable-matrix in the SDP, and C the number of constraints. We will compare the related computational complexity of the different upper bound variants in this section. Let ϕ be a CNF formula with n variables and m clauses.

In the Goemans-Williamson approach the size of the variable matrix is $n + 1$, and the number of constraints is also $n + 1$ implying a computational complexity

$$CP_{GW} = \mathcal{O}(n^2\sqrt{n})$$

The variant FG_m also has a variable matrix of size $n + 1$, but the number of constraints is now $n + 1 + m$ yielding

$$CP_{FG_m} = \mathcal{O}((n^2 + m)\sqrt{n})$$

d	OPT	#E SOS_p	Av SOS_p	Min SOS_p	#E FG_{ap}	Av FG_{ap}	Min FG_{ap}	#I
1.5	36	2	0.996438	0.989314	0	0.984777	0.980163	3
	37	27	0.999780	0.994716	8	0.996815	0.990767	29
2.0	47	5	1	1	1	0.990664	0.987512	5
	48	24	0.999098	0.994749	5	0.995868	0.988283	30
	49	45	0.999896	0.995205	27	0.998918	0.989843	46
2.5	57	0	0.994421	0.994421	0	0.990546	0.990546	1
	58	0	0.999722	0.999722	0	0.990159	0.990159	1
	59	10	0.998753	0.992482	0	0.993466	0.986547	15
	60	20	0.999471	0.996081	3	0.996686	0.991281	24
	61	41	0.999901	0.995850	26	0.999238	0.994349	42
	62	15	1	1	13	0.999833	0.997988	15
3.0	69	10	0.999827	0.998096	0	0.994511	0.990119	11
	70	16	0.999802	0.996639	1	0.997208	0.990343	17
	71	28	0.999909	0.997371	12	0.998443	0.992015	29
	72	29	0.999951	0.998523	17	0.999285	0.996044	30
	73	10	1	1	9	0.999829	0.998291	10
	74	3	1	1	3	1	1	3
3.5	78	0	0.993485	0.993485	0	0.983817	0.983817	1
	79	1	1	1	0	0.990783	0.990783	1
	80	3	0.999334	0.998357	0	0.994561	0.991726	6
	81	20	0.999460	0.996669	2	0.995872	0.989009	26
	82	22	0.999734	0.996179	4	0.997746	0.993598	24
	83	19	1	1	6	0.998670	0.995682	19
	84	13	1	1	12	0.999866	0.998259	13
	85	8	1	1	7	0.99985	0.999879	8
	86	2	1	1	1	0.999274	0.998544	2
4.0	90	5	0.999629	0.997774	0	0.992851	0.990062	6
	91	15	0.999755	0.997380	1	0.995239	0.987905	17
	92	28	0.999964	0.998966	6	0.998158	0.992194	29
	93	21	0.999919	0.998221	5	0.998694	0.995554	22
	94	12	1	1	6	0.999628	0.997897	12
	95	10	1	1	7	0.999941	0.999478	10
	96	4	1	1	4	1	1	4
4.5	100	0	0.998557	0.997645	0	0.990234	0.989394	3
	101	4	0.999535	0.998409	0	0.993362	0.989456	7
	102	18	0.999878	0.997691	0	0.995885	0.989874	19
	103	23	1	1	7	0.998316	0.994168	23
	104	15	0.999882	0.998112	3	0.998851	0.994588	16
	105	15	1	1	6	0.998962	0.997240	15
	106	13	1	1	12	0.999903	0.998735	13
	107	4	1	1	4	1	1	4
5.0	110	1	1	1	0	0.994511	0.994511	1
	111	8	0.999885	0.998964	0	0.994919	0.992738	9
	112	14	1	1	1	0.997640	0.994068	14
	113	24	1	1	2	0.997601	0.993335	24
	114	19	0.999999	0.999973	7	0.998725	0.993769	20
	115	11	1	1	5	0.999383	0.997228	11
	116	14	1	1	8	0.999580	0.996685	14
	117	4	1	1	3	0.999897	0.999586	4
	118	3	1	1	3	1	1	3

Table 4.3: 25 variables, MAX-2-SAT, bound ratios and frequencies of finding optimum

Hence, for fixed clause-variable density, FG_m has complexity $\mathcal{O}(n^2\sqrt{n})$. Using that the number of pairs of variables occurring in a same clause is of the same order as the number of clauses for 2-SAT, we have

$$CP_{FG_{4p}} = \mathcal{O}((n^2 + m)\sqrt{n})$$

In the SDP of FG_{ap} the size of the variable matrix is $n + 1$, and the number of constraints $n + 1 + 4p$ with the number of pairs p equal to $\frac{1}{2}n(n - 1)$. For the computational complexity of $CP_{FG_{ap}}$, we have

$$CP_{FG_{ap}} = \mathcal{O}((4n^2 + 2n + 2)\sqrt{n}) = \mathcal{O}(n^2\sqrt{n})$$

The variable matrix of FG_{pt} is of size $n + 1$, and the number of constraints equals $n + 1 + 2n(n - 1) + \frac{2}{3}n(n - 1)(n - 2)$ giving

$$CP_{FG_{pt}} = \mathcal{O}(n^3\sqrt{n})$$

SOS_p is applicable for both MAX-2-SAT and MAX-3-SAT, but with a slightly different complexity. The size of the variable matrix equals the size $|M|$ of the monomial basis. For MAX-2-SAT, the size of the basis is smaller than or equal to $1 + n + m$. The number of constraints is of $\mathcal{O}(|M|^2)$. Hence, for MAX-2-SAT, the computational complexity of CP_{SOS_p} is

$$CP_{SOS_p} = \mathcal{O}((n + m)^2\sqrt{n + m})$$

Note that for instances with a fixed clause-variable density d , i.e. $m = dn$, the computational complexity of FG_{ap} and SOS_p is of the same order for MAX-2-SAT. For MAX-3-SAT, the size of the monomial basis is at most $1 + n + 3m$ giving the complexity

$$CP_{SOS_p} = \mathcal{O}\left((n + 3m)^2\sqrt{n + 3m}\right)$$

Finally, SOS_{ap} has variable matrix size $n + 1 + \frac{1}{2}n(n - 1)$ and $n + 1 + \frac{1}{2}n(n - 1) + \frac{1}{6}n(n - 1)(n - 2)$ constraints. The number of constraints is obtained by observing that in $M^T SM = F_\phi - \alpha$ there are linear constraints for the constant, each single variable, each pair of variables and each triple of variables. Because the variable matrix is of size $\mathcal{O}(n^2)$ the complexity is

$$CP_{SOS_{ap}} = \mathcal{O}(n^5)$$

The variants SOS_t and SOS_{pt} , when applied to MAX-3-SAT, have the following complexities

$$CP_{SOS_t} = \mathcal{O}((n + m)^2\sqrt{n + m})$$

$$CP_{SOS_{pt}} = \mathcal{O}\left((n + 4m)^2\sqrt{n + 4m}\right)$$

4.4 SDP-based upper bounds for MAX-3-SAT

In contrast to the Goemans-Williamson and Feige-Goemans approaches the SOS-approach is directly applicable for MAX-3-SAT. Karloff and Zwick [84] present an algorithm based on semidefinite programming that guarantees a $7/8$ -approximation of MAX-3-SAT. Karloff and Zwick [84] prove that this is the case for satisfiable instances and provide strong evidence that it is also the case for unsatisfiable MAX-3-SAT instances. Zwick [156] completes the proof that the method is also a $7/8$ -approximation for unsatisfiable instances. Håstad [67] proved that for any $\epsilon > 0$ there does not exist a polynomial time $(\frac{7}{8} + \epsilon)$ -approximation algorithm for MAX-3-SAT unless $\mathcal{P} = \mathcal{NP}$. This result implies that the algorithm by Karloff and Zwick is as tight as possible for the complete class of MAX-3-SAT instances.

We will start with a short description of the semidefinite program of Karloff and Zwick for unweighted MAX-3-SAT. Literals are numbered from 1 to $2n$ in the order $X_1, \dots, X_n, \neg X_1, \dots, \neg X_n$. Let z_{ijk} be a boolean variable being 1 if the clause with literals i, j and k is satisfied and 0 otherwise. v_1, \dots, v_n are vectors in \mathbb{R}^{n+1} corresponding to the literals X_1, \dots, X_n , and v_{n+1}, \dots, v_{2n} correspond to the literals $\neg X_1, \dots, \neg X_n$. v_0 is a vector corresponding to FALSE. The program presented by Karloff and Zwick [84] is

$$\max \sum_{i,j,k} z_{ijk} \tag{4.4.1}$$

$$z_{ijk} \leq \frac{4 - (v_0 + v_i) \cdot (v_j + v_k)}{4} \tag{4.4.2}$$

$$z_{ijk} \leq \frac{4 - (v_0 + v_j) \cdot (v_i + v_k)}{4} \tag{4.4.3}$$

$$z_{ijk} \leq \frac{4 - (v_0 + v_k) \cdot (v_i + v_j)}{4} \tag{4.4.4}$$

$$z_{ijk} \leq 1 \tag{4.4.5}$$

$$v_{n+i} = -v_i \tag{4.4.6}$$

Just like in the Goemans-Williamson approach inproducts $v_i v_j$ are replaced by variables t_{ij} to obtain the SDP. The sum in (4.4.1) is taken over all i, j, k such that there is a clause with literals i, j and k . Constraint (4.4.6) implies that v_i corresponding to X_i must be the opposite of v_{n+i} corresponding to $\neg X_i$. It is easy to check that one of (4.4.2) to (4.4.4) forces z_{ijk} to 0 if the vectors take values corresponding to an assignment that does not satisfy the clause with literal i, j and k .

In this section, we prove that SOS_{pt} gives at least as tight upper bounds as the approach of Karloff and Zwick and present experimental results for a set of randomly generated MAX-3-SAT instances.

4.4.1 Karloff-Zwick inequalities vs monomials in SOS_{pt}

Theorem 4.4.1. *Each constraint of the type (4.4.2), (4.4.3) or (4.4.4) in the SDP of Karloff and Zwick can be represented as an inequality that states that a sum of squares is non-negative with respect to the monomial basis*

$$M = \{1, x_i, x_j, x_k, x_i x_j, x_i x_k, x_j x_k, x_i x_j x_k\}$$

Proof. In the SDP of Karloff and Zwick the z_{ijk} are variables having the interpretation of being 1 if the corresponding clause is satisfied, and 0 if not.

For the clause $X_i \vee X_j \vee X_k$, the expression

$$1 - \frac{1}{8}(1 - x_i)(1 - x_j)(1 - x_k) \quad (4.4.7)$$

is equivalent to z_{ijk} in the sense that is also 1 if the clause is satisfied, and 0 otherwise. Inequality (4.4.2) is for this clause equal to

$$1 - \frac{1}{4}(-1 + x_i)(x_j + x_k) \geq 1 - \frac{1}{8}(1 - x_i)(1 - x_j)(1 - x_k),$$

which is equivalent to

$$(1 - x_i)(1 - x_j)(1 - x_k) + 2(1 - x_i)(x_j + x_k) \geq 0$$

This can be simplified to

$$(1 - x_i)(1 + x_j + x_k + x_j x_k) \geq 0.$$

Hence we have to show that $(1 - x_i)(1 + x_j + x_k + x_j x_k)$ is a sum of squares modulo $I_{\mathcal{B}}$ invoking the designated monomials. This can be seen by

$$\begin{aligned} (1 - x_i)(1 + x_j + x_k + x_j x_k) &\equiv \\ \frac{1}{8}(1 - x_i + x_j + x_k - x_i x_j - x_i x_k + x_j x_k - x_i x_j x_k)^2 &\text{ modulo } I_{\mathcal{B}} \end{aligned} \quad (4.4.8)$$

The other constraints (4.4.3) and (4.4.4) can be dealt with analogously. This completes the proof. \square

The next corollary is a consequence of the above theorem. The reasoning is similar as in the proof of Theorem 4.2.2.

Corollary 4.4.2. *SOS_{pt} gives at least as tight upper bounds as the approach of Karloff and Zwick.*

n	d	SOS_{pt}	#E	SOS_p	#E	SOS_t	#E	#I
10	4.0	0.99997691	17	0.999486	9	0.988574	0	18
	5.0	1	56	0.999871	40	0.993319	1	56
	6.0	1	84	0.999987	75	0.995770	10	84
	7.0	1	98	0.999977	94	0.997125	23	98
	8.0	1	99	0.999998	98	0.997873	30	99
	9.0	1	100	0.999997	99	0.997672	23	100
15	4.0	0.99997251	9	0.999288	2	0.985297	0	10
	4.5	1	32	0.999806	15	0.989574	0	32
	5.0	0.99999802	55	0.999934	41	0.993318	0	58
	6.0	1	92	0.999984	83	0.996899	8	92
	7.0	0.99999959	99	0.9999902	98	0.997871	18	100
	8.0	0.99999954	98	0.99999606	96	0.998280	18	100
	9.0	0.99999677	99	0.99999623	98	0.998556	30	100

Table 4.4: 10 and 15 variables MAX-3-SAT, different SOS upper bounds of unsatisfiable instances

4.4.2 Experimental results for random MAX-3-SAT

We would like to compare the upper bound obtained by SOS_t , SOS_p and SOS_{pt} with the upper bound found by the Karloff-Zwick semidefinite program on randomly generated MAX-3-SAT instances with 10 and 15 variables and different densities. For the selected instances, the upper bound obtained by the Karloff-Zwick SDP is always equal to the number of clauses. Hence, in the remainder we will present only the results for SOS_t , SOS_p and SOS_{pt} .

In Table 4.4 the first column gives the number of variables, the second column the density, the third column the bound ratio for SOS_{pt} . Column 4 gives the number of instances for which the upper bound of SOS_{pt} equals the optimal MAX-SAT solution up to a given precision. Columns 5 to 8 give similar results for SOS_p and SOS_t . The last column gives the number of instances used. From Table 4.4 it is clear that SOS_{pt} comes very close to the optimal MAX-SAT solution for the instances of the size considered in the experiment. In fact, in most cases the SOS based upper bound equals the MAX-SAT solution value.

4.5 Experimental results for SOS_t and SOS_{pt} on (the decision variant of) 3-SAT

The SOS approach is not only useful as a method to approximate MAX-SAT, but can also be used to prove the unsatisfiability of an instance. If it returns an upper bound smaller than the number of clauses (minus some small value γ), the instance is definitely unsatisfiable. The value γ is necessary to compensate for numerical imprecisions.

cisions. In this section we use a set of randomly generated 3-SAT instances with 15 and 20 variables and densities varying between 4.0 and 5.0, 100 instances for each size and density.

For any of the unsatisfiable instances in the set of selected instances, we computed the upper bound for SOS_t . Secondly, we computed the upper bound of SOS_{pt} . We selected besides SOS_{pt} also SOS_t because SOS_t can be solved much faster than SOS_{pt} . Based on these upper bounds we can count the number of instances that is proved to be unsatisfiable, i.e. the number of instances for which the upper bound is smaller than the number of clauses (minus γ).

Anjos [5] proposed a new SDP relaxation for SAT, which significantly improved on earlier SDP relaxations, and can prove that a CNF-formula is unsatisfiable for formulas containing clauses of any length. Higher liftings are used to obtain the relaxation. The experiments in his paper show that for instances with up to 260 variables and 400 clauses satisfying assignments or proofs of unsatisfiability can be obtained. We compare the SOS approach with this rank-3-SDP relaxation for satisfiability on a set of unsatisfiable instances. The method by Anjos proves unsatisfiability by proving infeasibility of his SDP, which we will describe below.

Let P be the set containing $1, x_1, \dots, x_n$, all $x_i x_j$ such that variables X_i and X_j occur in a same clause and all $x_i x_j x_k$ such that variables X_i, X_j and X_k occur in a same clause. Let v be a column vector containing these monomials and $Y = vv^T$. Let I denote the set of indices of the variables in a monomial. $Y_{\emptyset, I} = \prod_{i \in I} x_i$. Define s_{ij} to be 1 if X_j is contained in clause i and -1 if $\neg X_j$ is contained in clause j . The rank-3-relaxation is to find a symmetric positive semi-definite $|P| \times |P|$ matrix Y with ones on the diagonal satisfying

$$s_{j1}Y_{\emptyset, \{i_1\}} + s_{j2}Y_{\emptyset, \{i_2\}} - s_{j1}s_{j2}Y_{\emptyset, \{i_1, i_2\}} = 1 \quad (4.5.1)$$

for each clause j of length two, with i_1 and i_2 the indices of the variables in clause j and

$$\begin{aligned} s_{j1}Y_{\emptyset, \{i_1\}} + s_{j2}Y_{\emptyset, \{i_2\}} + s_{j3}Y_{\emptyset, \{i_3\}} - s_{j1}s_{j2}Y_{\emptyset, \{i_1, i_2\}} - s_{j1}s_{j3}Y_{\emptyset, \{i_1, i_3\}} \\ - s_{j2}s_{j3}Y_{\emptyset, \{i_2, i_3\}} + s_{j1}s_{j2}s_{j3}Y_{\emptyset, \{i_1, i_2, i_3\}} = 1 \end{aligned} \quad (4.5.2)$$

for each clause j of length three, with i_1, i_2 and i_3 the indices of the variables in clause j , and furthermore the constraints

$$Y_{\emptyset, I_1} = Y_{I_2, I_3}, Y_{\emptyset, I_2} = Y_{I_1, I_3}, Y_{\emptyset, I_3} = Y_{I_1, I_2} \quad (4.5.3)$$

for each I_1, I_2 and I_3 such that I_1, I_2 and I_3 are contained in the set of indices of variables in some clause j and $I_1 \triangle I_2 = I_3$ (i.e. I_3 contains all elements that are in I_1 or I_2 but not in both). Hence, the SDP of Anjos adds a new variable (to be interpreted as Boolean) for each 'clause pair' and each 'clause triple'. Note that the size of the matrices in the Anjos' SDP and the size of the matrices in M_{pt} are identical. Also the computational complexity is of the same order as the one of SOS_{pt} .

We thank Miguel Anjos for providing us with the code of his relaxation

n	d	#unsats	SOS_t	SOS_{pt}	A3
15	4.0	10	6	10	10
	4.1	17	11	17	17
	4.2	19	11	19	19
	4.3	24	15	24	24
	4.4	24	20	24	24
	4.5	32	28	32	32
	4.6	35	32	35	35
	4.7	40	35	40	40
	4.8	45	43	45	45
	4.9	54	51	54	54
5.0	58	54	58	58	
20	4.0	19	0	19	7
	4.1	25	0	25	12
	4.2	30	0	30	19
	4.3	36	0	36	22

Table 4.5: 15 and 20 variables 3-SAT

Table 4.5 shows the results for the three approaches on the set of instances with 15 variables and 20 variables. The first column gives the density, the second one the number of instances out of the 100 generated instances that are unsatisfiable. The third, fourth and fifth column give respectively the number of instances proved unsatisfiable by SOS_t , SOS_{pt} and the rank-3-relaxation of Anjos.

From Table 4.5 we might conclude that SOS_{pt} is able to prove unsatisfiability up to larger sizes than the rank-3-relaxation of Anjos, and that the ability of SOS_t to prove unsatisfiability breaks down early. Now we want to give some impression about the computation times involved. Unfortunately, we were not able to run the rank-3-relaxation under the same computational environment as SOS_t and SOS_{pt} , because it involves MATLAB routines. Hence, the figures below only give a brief indication of the trade-off between quality and computational effort. For the instances with 15 variables and density 4.3, SOS_t needs on average 35.1 seconds, SOS_{pt} 549.9 seconds, and the rank-3-relaxation by Anjos needs on average 5.8 seconds.

4.6 Rounding procedures based on SOS_p and SOS_t

Goemans and Williamson present in their paper a rounding procedure for obtaining assignments that give a lower bound on the MAX-SAT solution. The performance guarantee obtained by this rounding procedure and the upper bound obtained by their SDP is 0.87856. Feige-Goemans [53] improve on this with their approach having a performance guarantee of 0.931. Lewin, Livnat and Zwick present a skewed rounding procedure yielding a performance guarantee of 0.940. In this section we present a

rounding procedure for finding lower bounds based on the solutions of SOS_p and SOS_t .

We describe our rounding procedure for SOS_p , but the procedure can be generalized to any monomial basis. This SOS-rounding takes as input the optimal solution S of SOS_p . It can be shown that such an optimal S has an eigenvalue 0. Next the procedure determines an orthogonal basis of eigenvectors V_1, \dots, V_N of the optimal matrix S corresponding to eigenvalue 0. These eigenvectors might be algebraically inconsistent. With this we mean that for example the entries corresponding to x_1 and x_2 might be positive while the entry corresponding to x_1x_2 might be negative.

The eigenvectors corresponding to eigenvalue 0 are the most relevant vectors because vectors with eigenvalue 0 correspond with solutions: if \hat{V} is an eigenvector of S corresponding to eigenvalue 0, $M^T S M = 0$ if M_k is replaced by the numerical value of \hat{V}_k . If such an eigenvector is algebraically consistent and boolean, it realizes the optimal MAX-SAT solution.

Next, we come to the description of our randomized rounding procedure. A random point $\lambda = (\lambda_1, \dots, \lambda_N)$ on the N -dimensional unit sphere is generated uniformly. We use the method by Knuth to generate uniformly distributed random points on the N -dimensional unit sphere [90], which uses standard normal distributed variables. The *sign* operator on a number x is defined to return 1 if $x \geq 0$, and -1 otherwise. Let P be defined as

$$P = \sum_{i=1}^N \lambda_i V_i. \quad (4.6.1)$$

The sign operator on a vector is defined as applying the sign operator on each individual entry. Let $SP = \text{sign}(P)$. SP_2, \dots, SP_{n+1} give an assignment for the variables X_1, \dots, X_n in the CNF-formula. The entry SP_1 corresponds to the first monomial in the monomial basis, being 1. Therefore, if $SP_1 = -1$ we reverse the assignment of the variables.

It can be proved that the above rounding procedure gives identical results as the rounding procedure of Goemans-Williamson, if the monomial basis M_{GW} is used. Preliminary experiments showed that using the optimal matrix of SOS_p and the rounding procedure with a uniformly distributed λ does not necessarily improve on the Goemans-Williamson approach. The reason for this is probably that the eigenvectors obtained are algebraically inconsistent (apart from the fact that their entries are not necessarily boolean). This might influence the quality of the rounding procedure negatively.

The idea to improve on this rounding procedure is that we want to try to get the entries in P corresponding to the products of variables as small as possible, to minimize the influence of the algebraic inconsistency as much as possible. Let V be the matrix containing V_1, \dots, V_N as columns, and p be the number of monomials of the form $x_i x_j$ in the monomial basis. The $p \times N$ matrix B is the matrix containing

We thank Etienne de Klerk for noticing this

the rows $n+2$ to $n+1+p$ of V corresponding to the monomials of degree 2. We want to find vectors λ such that the last p entries in P are relatively small.

Therefore, we start with computing an orthogonal basis of eigenvectors U_1, \dots, U_N and eigenvalues μ_1, \dots, μ_N of the matrix $B^T B$. Let $\mu_1 \leq \mu_2 \leq \dots \leq \mu_N$. We have $B^T B U_i = \mu_i U_i$. Hence, we have

$$\|B U_i\|^2 = (B U_i)^T (B U_i) = U_i^T B^T B U_i = \mu_i U_i^T U_i = \mu_i \|U_i\|^2 = \mu_i$$

and we can conclude $\|B U_i\| = \sqrt{\mu_i}$. The goal is that we want to find a λ such that $B\lambda$ is small. We uniformly generate a random vector w on the N -dimensional sphere. For each $i = 1, \dots, N$, we determine a so-called scaling factors ξ_i . This results in a new skewed vector \tilde{w} with $\tilde{w}_i = w_i \xi_i$. The λ from the linear combination (4.6.1) is now taken as

$$\lambda = \sum_{k=1}^N \tilde{w}_k U_k.$$

Note that

$$\|B\lambda\| \leq \sum_{k=1}^n \tilde{w}_k \sqrt{\mu_k} = \sum_{k=1}^N \xi_k w_k \sqrt{\mu_k}$$

In the experiments we used scaling factors ξ_i such that $\xi_i \leq \xi_j$ if $i < j$. For i with μ_i relatively large, we want to keep ξ_i small in order to keep $\|B\lambda\|$ small.

4.6.1 Skewed rounding on MAX-2-SAT instances based on SOS_p

For the experiments in this section, we generated 100 instances with any of the following number of variables and densities: 40 variables and density 3.0, 35 variables and densities 3.0 and 5.0, 30 variables and densities 3.0, 5.0 and 7.0 and 25 variables and densities 3.0, 5.0, 7.0 and 9.0. We selected instances of this size because the corresponding SDPs can be solved in reasonable time with current SDP solvers. For any of the instances, we solve SOS_p and apply SOS_p -based rounding with a uniform random vector and five weighted rounding procedures as described above. The three scaling factors we use are $\rho_i^2, \rho_i^5, \rho_i^{15}, \rho_i^N$ with $\rho_i = (1 - (\mu_i - \mu_1))$. Also we investigate scaling factors $2^{-(i-1)}$. In the first four the scaling is dependent on the size of the eigenvalues of $B^T B$. The last factor depends only on the ranking of the eigenvalues. Note that the first four are increasingly steeper. ρ_i^5 is selected for its good performance in preliminary experiments. The others are selected to study the effect of the 'steepness' of the scaling factors. Scaling factors ρ_i^5 seems to offer a good balance between the influence of the eigenvectors corresponding to small and larger eigenvalues. A much steeper function for the scaling factors is not very desirable because the vectors with larger eigenvalues get a scaling factor that is nearly zero and hardly contribute to the linear combination. The λ -vectors obtained are much more similar to each other. Consequently, the number of different possible assignments found is considerably reduced. As a consequence the chance of finding an optimal solution is lower. A flatter

n	d	S_R	ρ_i^2	ρ_i^5	ρ_i^{15}	ρ_i^N	$2^{-(i-1)}$	GW
40	3.0	250.94	287.85	356.33	530.83	484.63	533.27	143.98
35	3.0	333.64	384.02	471.09	652.95	586.54	642.95	188.98
35	5.0	515.09	531.57	559.19	662.64	600.75	733.81	122.49
30	3.0	383.09	429.26	503.82	661.97	578.46	655.95	209.06
30	5.0	495.13	513.21	522.98	669.68	588.05	731.90	166.45
30	7.0	624.12	635.49	651.34	712.95	704.30	800.34	131.95
25	3.0	504.26	562.66	643.72	835.16	749.41	787.17	287.04
25	5.0	544.65	567.7	609.92	756.12	662.17	677.11	236.32
25	7.0	630.55	645.64	668.18	753.04	689.81	821.34	196.39
25	9.0	749.62	757.55	769.40	821.81	781.94	871.82	175.51

Table 4.6: MAX-2-SAT, rounding on instances with different sizes and densities, frequencies of finding the optimum

function like, for example, ρ_i^2 , yields a larger diversity of λ -vectors and assignments but the number of times the (almost) optimal assignments are found is relatively low. For instance-wise comparison we implemented the Goemans-Williamson rounding on the Goemans-Williamson SDP. Each rounding procedure is run 1000 times.

Let S_R be the rounding procedure with a uniform random vector λ without any scaling.

Table 4.6 gives the average over the instances of the number of the times the optimum is found in the 1000 tries. This gives an indication for the number of tries that is necessary to find the optimum with high probability. Table 4.6 shows that the rounding procedures based on SOS_p find the optimum on average in about four times more out of 1000 tries than the approach of Goemans and Williamson.

Next, we define the observed performance guarantee of the procedures based on SOS_p for a particular instance as the average number of clauses satisfied in a try divided by the upper bound obtained by SOS_p . The observed performance guarantee of the Goemans-Williamson approach for an instance is defined as the average number clauses satisfied in a try divided by the upper bound obtained by the Goemans-Williamson approach. Table 4.7 gives for each set of instances the average over the instances of the observed performance guarantees.

Figure 4.1 shows the performance guarantees and the results for the Goemans-Williamson approach and the SOS_p approach for a typical random 2-SAT instance with 25 variables and density 9.0. From right to left the vertical lines give the optimum solution, which can be found for instances of this size by the algorithm of Borchers and Furman [20], the performance guarantee of the approach of Lewin, Livnat and Zwick [95], the performance guarantee of the method of Feige and Goemans [53] and the performance guarantee of the Goemans-Williamson approach [60]. The leftmost non-vertical graph represents the performance of the Goemans-Williamson rounding for this particular instance. Each point (x, y) of the graph indicates the fraction y of the tries for which the assignment found satisfies more than x clauses. The right-most

n	d	S_R	ρ_i^2	ρ_i^5	ρ_i^{15}	ρ_i^N	$2^{-(i-1)}$	GW
40	3.0	0.953	0.967	0.977	0.985	0.986	0.985	0.949
35	3.0	0.967	0.977	0.984	0.989	0.988	0.990	0.951
35	5.0	0.985	0.987	0.990	0.994	0.993	0.995	0.960
30	3.0	0.967	0.977	0.984	0.989	0.988	0.989	0.951
30	5.0	0.983	0.986	0.989	0.993	0.992	0.995	0.961
30	7.0	0.992	0.993	0.994	0.996	0.996	0.998	0.966
25	3.0	0.979	0.985	0.991	0.996	0.995	0.995	0.954
25	5.0	0.984	0.987	0.991	0.996	0.994	0.994	0.963
25	7.0	0.993	0.994	0.995	0.997	0.996	0.998	0.968
25	9.0	0.995	0.996	0.996	0.998	0.997	0.998	0.971

Table 4.7: MAX-2-SAT, rounding on instances with different sizes and densities, observed performance guarantee

non-vertical graph gives the performance of SOS_p with scaled rounding with scaling factors $2^{-(i-1)}$. Unfortunately, no actual implementations of the rounding procedures of Feige and Goemans and Lewin, Livnat and Zwick were available in order to get a more complete view here.

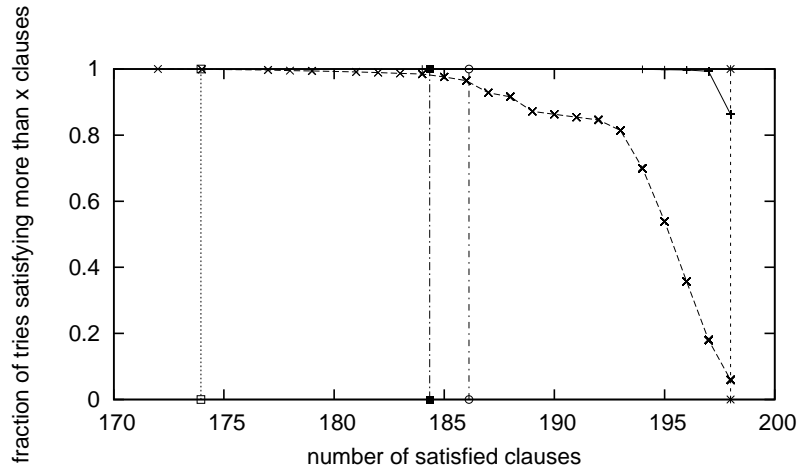


Figure 4.1: 25 variables, density 9.0, observed vs proven performance

From this subsection, we might conclude that the rounding procedure based on SOS_p with any of the scaling factors investigated has better observed performance guarantee and larger fraction of tries for which the optimum is found than the approach of Goemans and Williamson. The observed performance guarantee is on average better than the proven performance guarantees of the approaches of Feige and Goemans and Lewin, Livnat and Zwick. We observe that the steepness of the scaling factors is of influence, but not necessarily with identical impact on both aspects: the frequency of finding particular good solutions on one side and yielding high observed performance

on the other hand.

4.6.2 A rounding procedure for MAX-3-SAT based on SOS_t

In this section, we use a similar rounding procedure as in subsection 4.6.1, but instead of SOS_p we use SOS_t . In this rounding procedure the entries that we want to get relatively small are the entries corresponding to the monomials of degree 3. We chose SOS_t and not SOS_{pt} because SOS_t can be solved much faster than SOS_{pt} , and the size of the monomial basis is comparable to the size of the monomial basis of SOS_p in the MAX-2-SAT case. For the experiments on the rounding procedures for random MAX-3-SAT instances, we generated 100 unsatisfiable instances with 20 variables for any of the densities 4.0, 5.0, 6.0, 7.0, 8.0 and 9.0. We test the same variants of SOS-rounding as in the MAX-2-SAT case and compare the results for the Goemans-Williamson rounding applied on the SDP as presented by Karloff and Zwick [84]. We apply each rounding procedure a 1000 times on each instance.

Let SRT denote the rounding procedure with a uniform random vector λ without scaling.

Table 4.8 gives the average over the instances of the number of times out of the 1000 that the optimal number of clauses is attained. In this case, the observed performance guarantee for the procedures based on SOS_t is defined as the average number of clauses satisfied in a try divide by the upper bound obtained by SOS_t . The observed performance guarantee of the approach of Karloff and Zwick is defined as the average number of clauses satisfied in a try divided by the upper bound obtained by their SDP. Table 4.9 gives the average over the instances of the observed performance guarantees of the different approaches. Table 4.8 shows a trend break from density 7.0 to density 8.0. The average number of times the optimum is found decreases from density 4.0 up till density 7.0 and to density 8.0 there is an increase. The reason can be found in the number of eigenvectors corresponding to eigenvalue 0. Among the instances with density 8.0 and 9.0 are instances with a relatively small number of eigenvectors corresponding to eigenvalue 0, which is not the case for instances with smaller densities. The rounding procedures perform observably better when the number of eigenvectors corresponding to eigenvalue 0 is small. This might explain why the quality of the rounding starts performing much better from density 8.0 and higher.

Considering both tables shows a similar effect of the scaling as in Section 4.6.1, but considerably less pronounced. Note that the observed performance guarantee of the algorithm of Karloff and Zwick tends to its proven performance guarantee for larger densities. Notable is that the observed performance guarantee of the algorithm of Karloff and Zwick decreases with increasing density while the observed performance guarantee of the SOS-based procedures increases. This is mainly caused by the fact that the upper bounds tend to be better with larger densities using SOS_t .

To give some indication about the times needed to solve the SDPs, we remark that for the instances with 20 variables and density 7.0, the SDP of Karloff and Zwick takes on average about 6.2 seconds when solved by CSDP. On the same instances SOS_t takes

d	S_{RT}	ρ_i^2	ρ_i^5	ρ_i^{15}	ρ_i^N	$2^{-(i-1)}$	KZ
4.0	6.67	25.49	44.42	52.08	51.34	49.81	13.8
5.0	2.05	8.75	16.82	19.11	15.34	19.36	4.47
6.0	0.89	5.50	13.81	28.75	37.27	20.07	1.4
7.0	7.20	9.22	11.59	10.48	8.09	13.78	0.62
8.0	128.54	140.45	144.31	147.11	148.95	147.19	0.35
9.0	228.51	231.35	243.83	237.35	237.3	236.76	0.06

Table 4.8: MAX-3-SAT, 20 variables, average time optimum found

d	S_{RT}	ρ_i^2	ρ_i^5	ρ_i^{15}	ρ_i^N	$2^{-(i-1)}$	KZ
4.0	0.924	0.941	0.949	0.953	0.954	0.952	0.931
5.0	0.921	0.937	0.945	0.950	0.949	0.949	0.924
6.0	0.922	0.939	0.948	0.954	0.955	0.953	0.916
7.0	0.928	0.943	0.952	0.956	0.956	0.955	0.908
8.0	0.942	0.954	0.961	0.964	0.965	0.964	0.900
9.0	0.953	0.963	0.968	0.972	0.972	0.972	0.896

Table 4.9: MAX-3-SAT, 20 variables, observed performance guarantee

on average 2883.7 seconds on the same machine and with the same solver.

From the above observations we might conclude that the choice for the scaling factors depends on the goal: finding for as many instances as possible a relatively good solution or having a high probability of finding a particularly good solution in a try. The reason that scaling factors ρ_i^N perform relatively bad for these instances is that the number of eigenvectors corresponding to eigenvalue 0 tends to be quite large, yielding a very steep function such that only very few eigenvectors make a non-negligible contribution to the linear combination. Relatively flat scaling factors like, for example, ρ_i^2 yield λ -vectors that are relatively much different among each other. This leads to many different assignments in the tries. Among these assignment are very good ones and very bad ones, but the chance on a very good one is not very large. Steeper scaling factors like ρ_i^{15} , for example, give assignments that are not very different from each other and most are good. Because most assignments are quite similar one might miss the optimal solution in any try, although the assignment are on average very good.

Tables 4.6, 4.7, 4.8 and 4.9 show that is more difficult to find the optimal solution with the rounding procedure for MAX-3-SAT than for MAX-2-SAT for instances of the size considered.

The above instances are pure MAX-3-SAT instances. Because the upper bound found using the algorithm of Karloff and Zwick equals the number of clauses for all of these instances, we investigate a set of instances with mixed clause lengths. We present in Tables 4.10 and 4.11 the results for the algorithm of Karloff and Zwick and SOS_{pt}

C_1	C_2	C_3	KZ	S_{RT}	ρ_i^2	ρ_i^5	ρ_i^{15}	ρ_i^N	$2^{-(i-1)}$
10	10	60	0.975	0.98	0.982	0.985	0.992	0.992	0.995
10	15	55	0.97	0.972	0.976	0.981	0.989	0.988	0.991
10	20	50	0.977	0.98	0.983	0.986	0.992	0.99	0.994
10	5	65	0.975	0.983	0.985	0.987	0.993	0.993	0.996
15	5	60	0.981	0.982	0.984	0.987	0.992	0.991	0.995
5	10	65	0.963	0.983	0.984	0.987	0.993	0.992	0.996
5	20	55	0.969	0.982	0.985	0.987	0.993	0.992	0.995
5	25	50	0.974	0.981	0.983	0.986	0.992	0.992	0.995

Table 4.10: MAX-3-SAT, 20 variables, 80 clauses, observed performance guarantee

C_1	C_2	C_3	KZ	S_{RT}	ρ_i^2	ρ_i^5	ρ_i^{15}	ρ_i^N	$2^{-(i-1)}$
10	10	60	546.24	548.34	565.94	592.9	702.22	680.38	782
10	15	55	476.96	454.74	475.3	514.14	652.1	618.92	703
10	20	50	556.2	472.6	488.04	524.34	650.78	589.66	741.04
10	5	65	539.04	568.32	585.75	596	725.42	686.36	813.2
15	5	60	607.18	512.62	530.48	560.78	668.26	626.2	754.66
5	10	65	405.72	530.16	548.2	582.22	700.04	658.76	806.78
5	20	55	466.26	545.98	567.14	596.86	712.8	683.76	798.96
5	25	50	536.28	514.02	529.88	573.94	677.96	637.92	776.9

Table 4.11: MAX-3-SAT, 20 variables, 80 clauses, number of times optimum found

on a set of instances with 20 variables and 80 clauses of different lengths. The first column contains the number of clauses of length one, the second the number of clauses of length two and the third one the number of clauses of length three. Table 4.10 contains the observed performance guarantees and Table 4.11 the average number of times out of 1000 tries that the optimal solution is found. The fourth column contains the results for the algorithm of Karloff and Zwick, followed by the results for SOS_{pt} with unweighted rounding, rounding with respectively ρ_i^2 , ρ_i^5 , ρ_i^{15} and ρ_i^N . The last column contains the results for rounding with scaling factor $2^{-(i-1)}$.

From Table 4.10 and 4.11 it is clear that the performance guarantee of any of the SOS_{pt} -variants is better than Karloff and Zwick's. The variant with the best performance also obtains the optimal solution more often than Karloff and Zwick's method.

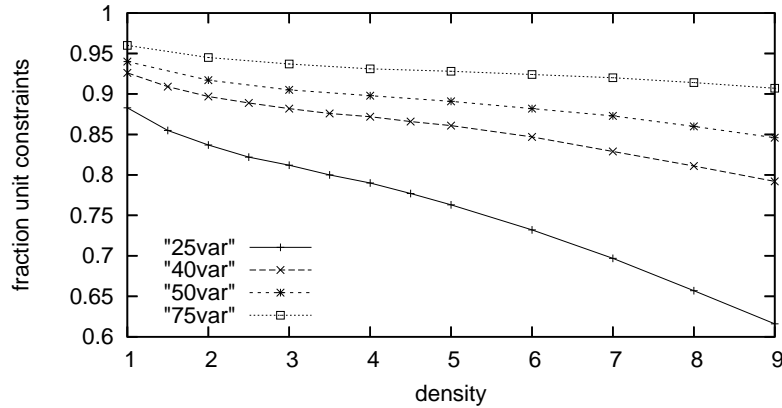


Figure 4.2: Fraction of unit constraints, 2-SAT

4.7 Fraction of unit constraints in SOS_p , SOS_t and SOS_{pt}

In this section we investigate the types of constraints in SOS_p and show that many of the constraints are of the form $s_{ij} + s_{ji} = c$ for some constant c . Because of the symmetry of the matrix S , these constraints simply fix the two matrix entries concerned. SDP-solvers that make use of the large fraction of these 'unit' constraints in an efficient way might be able to solve these SDPs much faster than current SDP-solvers. The results show that efforts should be made to design such solvers.

We generated random 2-SAT instances with any of the densities 1.0, 2.0, 3.0, ..., 9.0 and with 25, 40, 50 and 75 variables. For any of these densities and sizes, we determined the average percentage of unit constraints of the type $s_{ij} + s_{ji} = c$ in the corresponding SDP SOS_p .

Figure 4.2 illustrates that the fraction of unit constraints increases with problem size. The percentage of unit constraints decreases with increasing density for fixed size.

A small-scale experiment with randomly generated unsatisfiable 3-SAT instances with 20 variables and density 4.0 and density 5.0 showed that for these instances on average respectively 55.4% and 53.1% of the constraints in SOS_{pt} are unit constraints. Analogously to the 2-SAT case this percentage is expected to be larger for larger instances. A small set of instances with 40 variables and density 4.25 having 76.8% unit constraints illustrates this expectation to be valid. The percentage of unit constraints in SOS_t is considerably larger. For the same set of instances, SOS_t has on average 95.9% unit constraints.

4.8 Counterexamples to uncovered cases of Hilbert's Positivstellensatz?

The theory discussed in the introduction might be used to try to find a counterexample to one of the uncovered cases of Hilbert's Positivstellensatz. Note that there are finitely many different k -SAT CNF-formulas with n variables for fixed k . We showed in Section 4.1 that a formula ϕ is unsatisfiable if and only if there exists an $\epsilon > 0$ such that $F_\phi(x) \geq \epsilon$ for each $x \in \mathbb{R}^n$. Let Φ_n^k be the set of unsatisfiable k -SAT formula with n variables. Let ϵ_n^k be defined as

$$\epsilon_n^k = \min_{\phi \in \Phi_n^k} \left\{ \min_x F_\phi(x) \right\} \quad (4.8.1)$$

ϵ_n^k is always strictly positive. Practically, it is not possible to determine ϵ_n^k because it is the minimum over the exponentially many minima of all unsatisfiable k -SAT instances with n variables. Even finding the exact minimum for a given ϕ is not doable with current software.

Definition 4.8.1. *An unsatisfiable k -SAT formula ϕ with n variables is a CUC if $F_\phi - \epsilon_n^k$ is not a sum of squares.*

Alternatively, we can conclude that an unsatisfiable k -SAT formula ϕ is a CUC if the program

$$\begin{aligned} & \max \alpha & (4.8.2) \\ & \text{s.t. } F_\phi - \alpha \text{ is a SOS} \\ & \alpha \in \mathbb{R} \end{aligned}$$

returns an α smaller than or equal to ϵ_n^k . To decide this a semidefinite programming solver is necessary that returns a solution with precision at most ϵ_n^k .

In some small-scale experiments we compute the maximum of program (4.8.2) for any of the considered instances with the SDP solver CSDP [19]. We selected this solver because its precision turned out best among a small set of solvers we tried. We generated a set of 2-SAT instances with 10 variables with a density varying between 1.0 and 3.0. For all 1500 sample instances except two, the SDP (4.8.2) gives solutions such that any α corresponding to a satisfiable instance of a particular density is smaller than any α corresponding to an unsatisfiable instance of the same size and density although this difference may be very small. This implies that the considered satisfiable and unsatisfiable 2-SAT formulas are almost perfectly separated by their value of α . The precision of the solver is not sufficiently adequate because for some of the satisfiable instances, it returns a positive α , which is not allowed based on Theorem 1. Based on these experiments there is no reason to suppose that polynomials of degree 4 coming from unsatisfiable 2-SAT formulas may yield the desired counterexamples. On the other hand, due to the numerical imprecision the opposite cannot

be concluded either. Only much more accurate optimization methods, or much more accurate implementations of SDP-algorithms, could result in a more final conclusion.

In a small-scale experiment with 100 randomly generated 3-SAT instances with 10 variables and density 4.0 and 100 instances with density 5.0, we obtained similar results. We found an almost perfect separation of satisfiable and unsatisfiable instances based on the value of α but again no final conclusion can be made whether unsatisfiable random 3-SAT instances may provide CUCs. Although the experiments described give an unsatisfactory result concerning CUCs, they certainly indicate that the SOS approach has strong separating power regarding satisfiability. Because k -SAT is known to be \mathcal{NP} -complete (for $k \geq 3$), and the construction of the monomial basis M_ϕ using the Newton polytope for F_ϕ can be carried out in polynomial time, we can conclude with the following theorem:

Theorem 4.8.2. *If $\max \{ \alpha | F_\phi - \alpha = M_\phi^T S M_\phi, S \succeq 0 \} > 0$ can be decided in polynomial time (open) and $\mathcal{NP} \neq \mathcal{P}$, infinitely many CUCs exist for each $k \geq 3$.*

Corollary 4.8.3. *Under the same conditions, infinitely many polynomials of degree $2k$ ($k \geq 3$), coming from unsatisfiable instances, are non-negative but not sums of squares.*

4.9 Conclusions

In this paper, we compare the SOS (Sums Of Squares) approaches with existing upper bound and rounding techniques for the MAX-2-SAT case of Goemans and Williamson [60] and Feige and Goemans [53] and the MAX-3-SAT case of Karloff and Zwick [84], which are based on Semidefinite Programming as well. We prove that for any of these algorithms there is a SOS-based counterpart providing upper bounds at least as tight, but observably tighter in particular cases.

We conclude that a combination of the rounding schemes of Goemans and Williamson and of Feige and Goemans with the appropriate SOS based upper bound techniques proposed, provides polynomial time algorithms for MAX-2-SAT having a performance ratio guarantee at least as good as the ones proven by Goemans and Williamson and Feige and Goemans, but observably better in particular cases. A similar conclusion can be drawn with respect to the Karloff and Zwick algorithm for MAX 3-SAT. Also, the experiments with the newly proposed randomized rounding schemes for SOS_p and SOS_t seem promising.

Chapter 5

Forbidden color problem instances

In this chapter, we describe a method to generate satisfiability instances that are on average more difficult to solve than satisfiability instances coming from graph- k -coloring instances and give experimental evidence of the fact that these instances are indeed more difficult to solve for two state-of-the-art solvers. The forbidden color problem studied in this chapter has practical applications in several fields. Formulations of crew rostering or class scheduling, for example, might have the structure of a forbidden color problem. Coloring a node corresponds in the first case to assigning a crew member to a flight and in the second case to assigning courses to available time slots. Forbidden colors represent for example that certain crew members are not allowed on certain flights due to labor regulations or that persons cannot be assigned to overlapping time slots. Each scheduling or rostering problem involves a number of different resources like for example truck drivers, pilots, planes or rooms. The analysis of the behavior of the model with respect to the forbidden color percentage might help to discover how many of each of the resources are desirable.

Studying the behavior of structured problem instances with respect to the forbidden color percentage gives insight in the sensitivity of the problem instance. If an instance gets already unsatisfiable after forbidding only a few colors, it has very few solutions and is very sensible for local disturbances. Instances remaining satisfiable even after forbidding lots of colors are very robust with respect to local disturbances but have the disadvantage that there are many more resources than you need which makes it too expensive. By studying the sensitivity of the instance, it is possible to find an amount of any of the resources such that the model is not too sensitive for local disturbances but also not too expensive.

If for example, rostering or scheduling problems are modeled as graph coloring instances it is very important information to know how sensitive your model is for local disturbances. It would be very much desirable to have a model that is not too sensitive but also not too robust for disturbances. In that case, you can without too much problems chose an alternative schedule or roster if necessary but there are not too many resources which make it unnecessarily expensive. If you know your model is very sensitive to local disturbances, you might choose add increase some of the resources

to obtain a more robust model. If the model is too robust and hence, too expensive, the analysis with respect to the forbidden color percentage might help you to choose which of the resources to decrease. With a forbidden color based approach one can get this insight in the sensitivity of the model with respect to different resources.

Studying the behavior of instances with respect to the forbidden color percentage is somewhat related to the problem described in Gomes and Shmoys [63]. The problem in that paper is to check if it possible to completely fill a partially filled latin square. The larger the partial filling the lower the chance that the square can be filled completely. The authors analyze how the fraction of solvable latin squares is correlated with the size of the partial filling. Some planning problems can be formulated as latin squares, for example if n people must do each n different tasks on a day, and each task can be done by only one person at a time.

5.1 Transforming graph coloring to satisfiability

Let $G = (V, E)$ be a graph with V the set of nodes and E the set of edges. Let $|V|$ be the number of nodes in G . The *graph coloring problem* is to find the minimal number of colors such that each node in V can be colored and for each edge (i, j) , nodes i and j do not have the same color. Two nodes i and j are adjacent if there is an edge $(i, j) \in E$. The *node-edge density* of a graph is the number of edges divided by the number of nodes. Graph- k -coloring is the question whether a graph $G = (V, E)$ can be colored with k colors such that no two adjacent nodes have the same color.

The graph coloring problem is \mathcal{NP} -hard in the strong sense. A graph- k -coloring instance can be transformed into an instance of the satisfiability problem in polynomial time in the following way. For each node i , boolean variables x_{ij} , $j = 1, \dots, k$ are introduced, where x_{ij} equals 1 if node i has color j and 0 otherwise. There are $|V|$ so-called node clauses, expressing that each node has at least one color. For example the node clause for node i is

$$x_{i1} \vee \dots \vee x_{ik},$$

which states that node i should have at least one of the colors $1, \dots, k$. Additionally, the CNF-formula contains for each edge $(i, j) \in E$ k edge clauses of the form

$$\neg x_{ic} \vee \neg x_{jc}, c = 1, \dots, k,$$

indicating that two adjacent nodes i and j do not have the same color c , $c = 1, \dots, k$.

This formulation allows that nodes are colored with multiple colors, but a solution to this problem can easily be transformed into a solution with only one color per node. For each node with multiple colors, one of them can be selected arbitrarily.

5.2 Preliminary definitions

In the remainder of this chapter, we will call the transformation of graph- k -coloring into satisfiability Graph- k -SAT. In this chapter we present a method for generating so-

called forbidden color instances based on Graph- k -SAT and show experimentally that these instances are on average more difficult to solve for state-of-the-art satisfiability solvers than the instances they come from.

Graph- k -SAT shows strong threshold behavior with respect to the node-edge density of the graph. We show in our experiments that forbidden color instances coming from graphs with a fixed node-edge density near this density threshold show a weak phase transition with respect to the forbidden color percentage.

The Graph- k -SAT instance on which a forbidden color instance is based is called its *source instance*. A different parameter, scaled density, is an additional indicator of the difficulty of instances. Let d^* be the clause-variable density of a forbidden color instance after propagating all unit clauses. The *scaled density* of a forbidden color instance is defined as its d^* divided by the clause-variable density of its source instance. Among instances with a fixed forbidden color percentage, instances with a higher scaled density are more difficult to solve. This indicates that if a suitable forbidden color percentage and scaled density are chosen based on a particular Graph- k -SAT instance, it is possible to generate randomly forbidden color instances that are on average more difficult than the source instance, which already originates from an \mathcal{NP} -hard problem. The *scaled run time* is defined as the time needed to solve the forbidden color instance divided by the time needed to solve the source instance.

5.3 A generator for forbidden color instances

In this section, we will discuss how to randomly generate forbidden color instances. A Graph- k -SAT instance will be extended with ‘forbidden color clauses’. These clauses are of the form $\neg x_{ic}$, expressing that node i is not allowed to be colored with color c . The forbidden color clauses are unit clauses and hence can be propagated by unit propagation. This reduces the size of the formula but can make the formula more difficult as we will show later on.

We generate a forbidden color instance by randomly selecting combinations of nodes and forbidden colors uniformly. Let p be the so-called forbidden color percentage. $p\%$ of the possible assignments of colors to nodes are forbidden, and the corresponding unit clauses are added to the Graph- k -SAT instance. This implies that for a Graph- k -SAT instance with n variables, $\frac{pn}{100}$ unit clauses are added to the CNF-formula of the source instance to obtain a forbidden color instance with a forbidden color percentage p . One result can be that a node is not allowed to have any of the colors, which leaves the formula unsatisfiable. Fortunately, this happens only rarely if the forbidden color percentage is small enough (for Graph-3-SAT, smaller than 5%).

5.4 Experiments on forbidden color instances based on random Graph-3-SAT instances

In this section we show that on average forbidden color instances coming from Graph-3-SAT instances are more difficult than their source instances. This effect is more prominent for larger instances.

5.4.1 Experiments on small instances

In this subsection we present some small-scale experiments to give an idea of the phenomena observed for forbidden color instances based on randomly generated Graph-3-SAT instances. We selected instances with 250 nodes and 572 edges. Graph-3-coloring instances with 250 nodes have about 50% satisfiable instances if the number of edges is 572. These instances tend to be difficult compared to instances with more or less edges. In the experiments in this subsection, we use five randomly generated satisfiable instances with 250 nodes and 572 edges as source instances to illustrate our findings. It is essential to select only satisfiable source instances to get a mix of satisfiable and unsatisfiable forbidden color instances. Adding forbidden color clauses to unsatisfiable instances only yields instances that are even more unsatisfiable and hence likely to be easier to solve for satisfiability solvers.

For each of the five different randomly generated satisfiable graph-3-coloring instances with 250 nodes and 572 edges, we generated 1000 instances with forbidden color percentage 1 or 2. The percentage for each source instance was chosen such that the percentage of satisfiable instances is closest to 50%. For three instances, this is 2% (273, 632 and 654 out of the 1000 instances are satisfiable, respectively). For the other two instances, the percentage is 1%, and 733 and 554 instances out of 1000 are satisfiable, respectively. For each of the forbidden color instances obtained in this way we determined the scaled density. All source instances and forbidden color instances were solved by the satisfiability solvers *chaff* and *OKsolver*.

The forbidden color instances were divided in eleven classes depending on their scaled density. The first class contains instances with a scaled density between 0.92 and 0.93, the second one instances with scaled density between 0.93 and 0.94 etc. Figure 5.1 shows the average scaled run time for each of these classes of instances for both the *OKsolver* and *chaff*. This illustrates that on average instances with a larger scaled density tend to be more difficult to solve for both solvers. The first picture in this figure shows that, on average forbidden color instances from one of the source instances in the class with the largest scaled densities take almost six times as much computation time than the source instance for *chaff*. For two of the five instances, we were not successful in generating a class of forbidden color instances with a particular scaled density that on average was more difficult to solve than its source instance for *chaff*.

For the *OKsolver*, for one of the source instances, forbidden color instances in the class with the largest scaled densities were even more than ten times as difficult as their

source instance. For this solver there was a scaled density for each source instance such that instances in the class with this scaled density are more difficult to solve than their source instance. This shows that it is possible to generate forbidden color instances that are considerably more difficult to solve than their source instances. From Figure 5.1 it is clear that the most difficult instances for both solvers are in the classes with the largest scaled density.

We next illustrate that by randomly generating the forbidden color instances we get a substantial number of instances in these two classes. The first graph in Figure 5.2 gives for each of the five selected source instances the number of forbidden color instances out of 1000 with a scaled density in each of the classes. This shows that it is possible to generate sufficient forbidden color instances with a scaled density in the two classes with largest scaled density. The second graph in Figure 5.2 gives the fraction of satisfiable instances in each class. This shows how fast the fraction of satisfiable instances increases with scaled density.

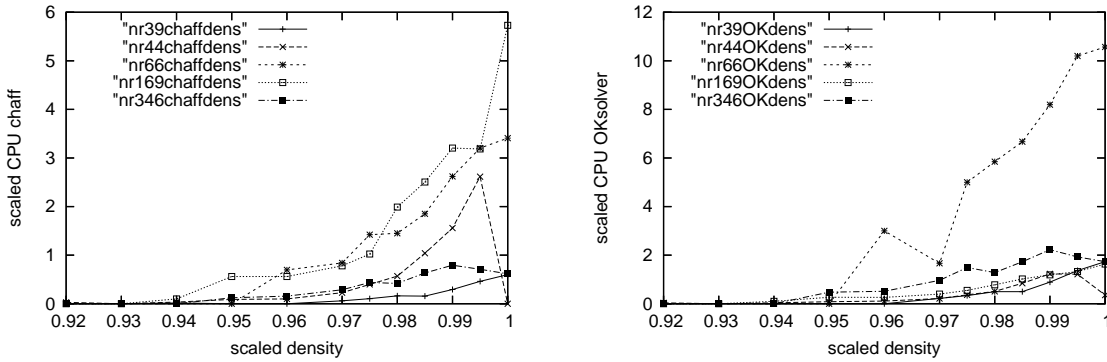


Figure 5.1: Graph coloring, 250 nodes

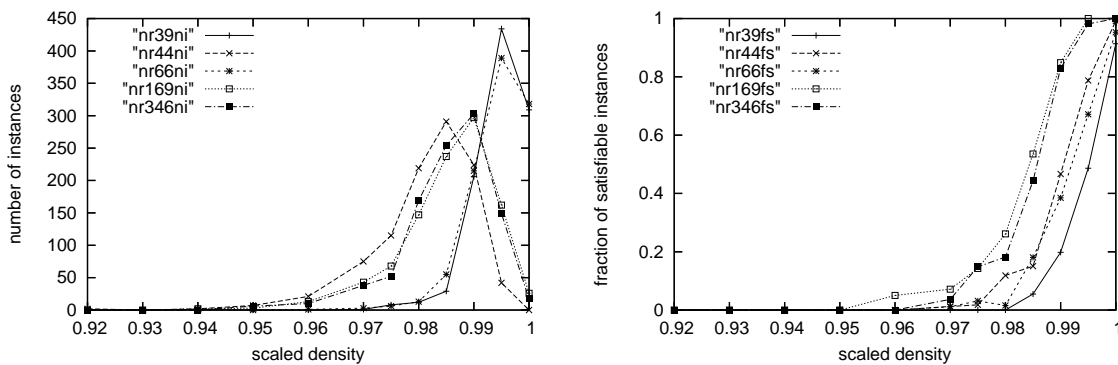


Figure 5.2: Graph coloring, 250 nodes

Figure 5.3 gives for a number of different forbidden color percentages the average computation time of chaff for forbidden color instances coming from each of the five source instances. The first picture shows the satisfiable instances which are the most

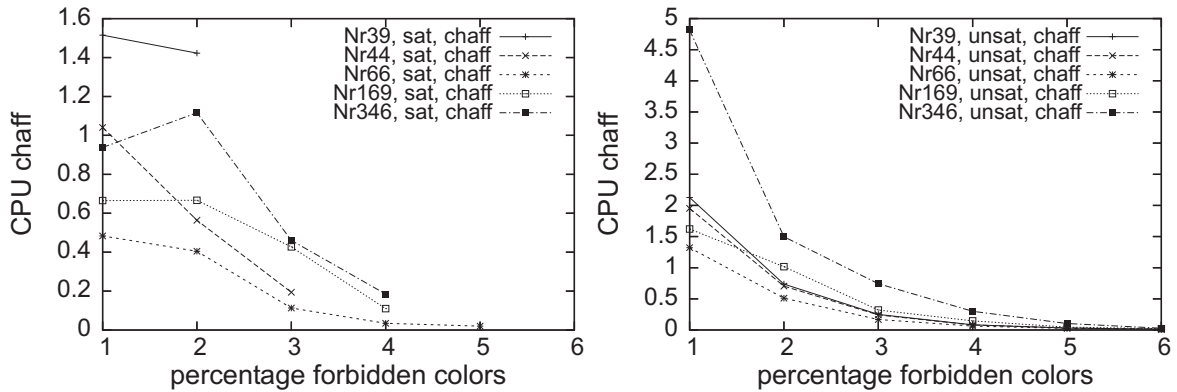


Figure 5.3: Graph coloring, 250 nodes

difficult for chaff, with 1 or 2% forbidden colors. For some of the source instances only the averages for the smaller forbidden color percentages are shown, because the set of generated forbidden color instances for the larger percentages did not contain any satisfiable instances for some of the source instances. The second picture shows that the unsatisfiable instances get easier with increasing forbidden color percentage. This fact is intuitively clear because forbidding more colors provides more possible ways to prove unsatisfiability.

5.4.2 Scaling to larger Graph-3-SAT instances

In this subsection we extend the interesting observations from the previous subsection to Graph-3-SAT instances with more nodes and edges and larger sets of source instances. We randomly generated 100 satisfiable Graph-3-SAT instances with 300 nodes and 690 edges. For each of these source instances, we generate 100 forbidden color instances for each of the forbidden color percentages 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 2.0 and 3.0. All source and forbidden color instances are solved with the OKsolver and chaff.

In Table 5.1 we summarize the results for this huge set of 80,000 forbidden color instances. In the first column we give the forbidden color percentage p . The second one, labelled 'sats', gives the number of source instances for which the number of satisfiable instances is closest to 50% for the corresponding forbidden color percentage. From this column it might be concluded that in most cases forbidden color percentages between 1% and 2% give an equal mix of both satisfiable and unsatisfiable instances. As graph-3-coloring instances at a node-edge density close to the density with 50% satisfiable instances tend to be relatively difficult to solve compared to instances with other densities, one might expect forbidden color instances with a forbidden color percentage where about 50% of the instances is satisfiable to be relatively difficult to solve. The column 'OKm' gives the number of source instances providing the most difficult instances for the OKsolver among the different forbidden color percentages. Note that quite a few of the source instances have the most difficult instances at a

p	sats	OKm	OKtimes	OKmsat	OKtimessat	chaffm	chafftimes	chaffmsat	chafftimessat
0.25%	0	2	2.8673	4	2.8123	7	2.1724	16	2.059
0.5 %	0	16	3.9343	18	3.4407	17	2.8187	20	2.607
0.75%	9	17	4.2604	13	3.6085	16	3.0356	7	2.373
1.0 %	26	20	4.1965	20	3.5633	16	3.0621	18	2.324
1.25%	15	24	4.0395	14	3.3971	16	2.7989	11	2.191
1.5 %	22	8	3.4594	12	3.0885	15	2.3672	16	1.993
2.0 %	26	13	2.5169	16	2.4959	13	1.6265	11	1.536
3.0 %	2	0	1.4138	3	1.7563	0	0.9999	1	1.067

Table 5.1: Graph-3-coloring with 300 nodes and 690 edges

lower forbidden color percentage than the one with about 50% satisfiable instances. The column 'OKtimes' indicates how many times more difficult to solve the forbidden color instances are on average than their source instance. For example, the 2.86725 in the first row indicates that forbidden color instances with $p=0.25\%$ take the OKsolver on average 2.86725 times as much computation time as their source instance.

Columns labelled 'chaffm' and 'chafftimes' give the same type of results for chaff. The columns 'OKmsat', 'OKtimessat', 'chaffmsat' and 'chafftimessat' give the same type of results but restricted to the satisfiable forbidden color instances. This illustrates that even the satisfiable forbidden color instances are more difficult than their source instances. For the OKsolver, satisfiable forbidden color instances take on average up to 3.6085 times more time to solve than their source instances if the forbidden color percentage is chosen to be 0.75%. For chaff, this difference is somewhat less, with 2.607 times as difficult for a forbidden color percentage of 0.5%.

From Table 5.1 it is directly clear that on average for any of the forbidden color percentages the forbidden color instances are more difficult to solve than their source instances for both solvers. For the OKsolver, instances with a forbidden color percentage of 0.75% are most difficult: on average 4.26041 times as difficult as the source instance. For chaff, instances with 1.0% forbidden colors are with 3.06212 times as difficult as the most difficult instances.

Table 5.2 gives the same type of results restricted to satisfiable instances as Table 5.1 but for larger graphs with 350 nodes and 805 edges. For the OKsolver, satisfiable instances with a forbidden color percentage of 1.0% are on average 16.539 times as difficult to solve as their source instance. Satisfiable instances with 0.75% forbidden colors are 17.223 times as difficult as their source instance for chaff. Table 5.2 illustrates that forbidden color instances with 350 nodes are considerably more difficult compared to their source instances than instances with 300 nodes.

For the set of 80,000 instances with 350 nodes and 805 edges, Table 5.3 contains in the first column the forbidden color percentage. The second column gives for the OKsolver the percentage of instances with the given forbidden color percentage that is more difficult to solve than its source instance. The third column gives the percentage of instances that is at least ten times as difficult to solve as the source instance. Table 5.4 gives the same type of results, but in this case related to the scaled density.

p	sats	OKmsat	OKtimessat	chaffmsat	chafftimessat
0.25%	0	4	11.453	10	14.167
0.5%	0	10	14.942	13	16.908
0.75%	14	15	16.092	13	17.223
1.0%	19	24	16.539	14	16.469
1.25%	18	19	15.224	22	14.651
1.5%	30	9	12.764	11	11.815
2.0%	19	19	9.521	17	9.056

Table 5.2: Graph-3-coloring with 350 nodes and 805 edges

p	P_1	P_{10}
0.25	47.75%	6.87%
0.5	56.29%	13.28%
0.75	60.23%	16.91%
1.0	63.89%	19.94%
1.25	65.31%	22.52%
1.5	65.41%	23.45%
2.0	65.05%	23.94%

Table 5.3: Graph-3-coloring, 350 nodes, with OKsolver

Combining a forbidden color percentage and a scaled density giving a large fraction of difficult instances is likely to give a class of instances that on average is even more difficult to solve on average.

5.5 Structured graph coloring instances

In this section, we consider a number of different structured graph coloring instances from the Graph Coloring Competition in 2002 and experimentally show that their behavior in terms of the forbidden color percentage is very different. For each instance we selected the number of colors k to be the smallest number of colors such that the graph is k -colorable.

For Figure 5.4 we selected two different types of instances, ash and DSJC. The ash instances originate from Hossain and Steihaug [77], [78]. These instances originate from the numerical estimation of the Jacobian matrix of mathematical derivatives of a function $f : \mathbb{R}^p \rightarrow \mathbb{R}^q$. The graph coloring problem is used to determine which entries in the Jacobian matrix are non-zero. DSJC [81] are standard (n,p) -random graphs. The random graph $G(n, p)$ is a graph where each possible edge between the nodes in V is chosen independently with probability p .

For forbidden color percentages between 2% and 14%, we determined the fraction

SD	P_1	P_{10}
2.6	64.26%	19.11%
2.61	66.62%	22.97%
2.62	68.02%	22.24%
2.63	65.49%	20.91%
2.64	59.91%	17.08%
2.65	55.88%	15.30%
2.66	51.10%	10.05%

Table 5.4: Graph-3-coloring, 350 nodes, with OKsolver

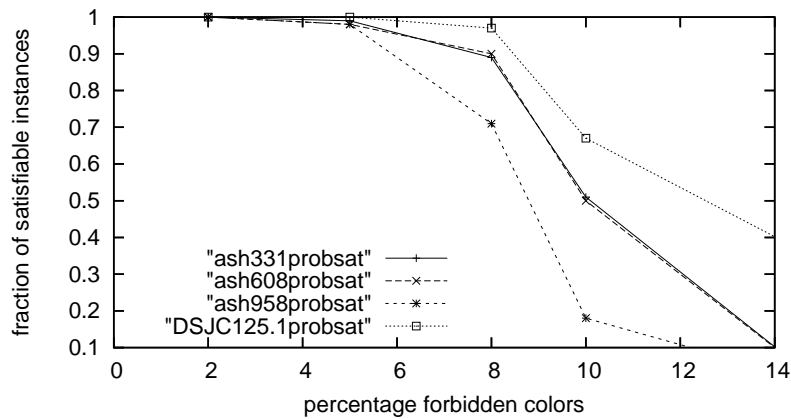


Figure 5.4: Structured graph coloring instances

of satisfiable instances. For the ash331 and ash608 instances, the forbidden color percentage with about 50% of the instances being satisfiable is 10%, for ash958 this is 9% and for DSJC125.1 11%. Note that these percentages are much larger than for the random Graph-3-SAT instances, where the point with about 50% satisfiable instances was at about 1% to 2% forbidden colors. The reason for this is that in the random graph-3-colorings the expected degree of each node is equal. For the structured instances, there are many nodes with a low degree. For nodes with a low degree, the influence of forbidding a color is less than forbidding a color for a high-degree node. A node with high degree is adjacent to many nodes and cannot have the same color as any of these nodes, implying that only a few colors are left to choose from. Nodes with a low degree have less constraints on their color.

Figure 5.5 illustrates how differently structured graph- k -coloring instances behave with respect to forbidden color percentage. The forbidden color coming from the Anna instances, for example, are still all satisfiable for a forbidden color percentage of 40%. The forbidden color instances coming from the le450-5 instance are already all unsatisfiable with a forbidden color percentage of only 1%.

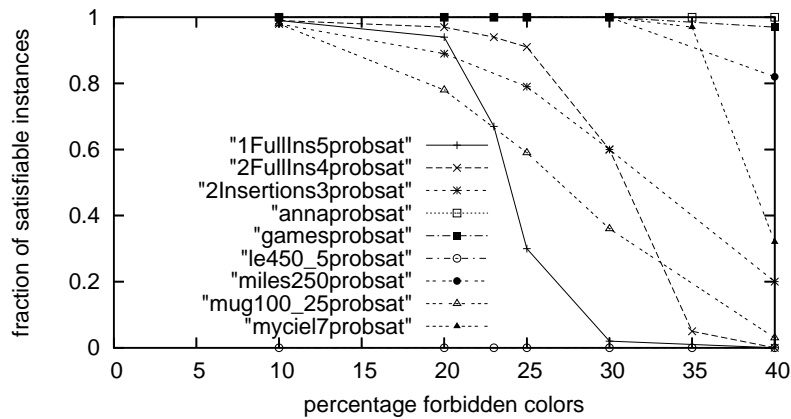


Figure 5.5: Structured graph coloring instances

5.6 Conclusions

In this chapter we presented a method to generate forbidden color instances based on Graph- k -SAT instances. On average, forbidden color instances are more difficult for state-of-the-art solvers than the Graph- k -SAT instances they originate from. The difficult satisfiability instances obtained in this way might be used for comparing performance of satisfiability solvers. If forbidden color instances are generated randomly, a sufficient number of difficult instances is found. It is notable that a large number of the satisfiable forbidden color instances found are quite difficult to solve because usually, satisfiable graph- k -coloring instances are very easy to solve compared to unsatisfiable ones. Furthermore, our experiments showed that structured problem instances have very different behaviour with respect to the forbidden color percentage.

Chapter 6

Conclusion

This thesis presents an algorithm to compute so-called k -level reserve Horn fractions. We presented evidence for the fact that the k -level reverse Horn fraction may play a similar (albeit weaker) role as the density parameter in separating satisfiable random $\{3\}$ -SAT instances from unsatisfiable ones. If it is impossible to discriminate using the density parameter, the k -level reverse Horn fraction seems suitable. From a statistical point of view it is favorable to combine weak separators. The k -level reverse Horn fraction can be combined with other weak separators as, for instance, discussed in [10] and [145]. It is remarkable that the k -level reverse Horn fraction indeed has weakly separating properties because random $\{3\}$ -SAT instances start with having expected reverse Horn fraction equal to 0.5, and hence seem rather insensitive with respect to this parameter. Performances of various complete and incomplete state-of-the-art solvers seem to be correlated with the k -level reverse Horn fraction as well, although any link to or use of reverse Horn clauses in these solvers is absent. Some theoretical back-up for the correlations observed come from [111] and [116]. In the first paper, a relation between Horn fraction and the size of "backdoor" sets is established. The size of these sets is correlated to computational effort of DLL-based solving techniques. The second paper shows that larger Horn fractions yield lower complexity worst case bounds.

Currently, the algorithm aHS which is used to compute the k -level reverse Horn fractions spends most of the time in solving the linear programs. The considerable amount of time necessary for solving the linear programs limits the size of the instances and the value of k for which the k -level reverse Horn fractions can be computed. It could be interesting to investigate the strongness of the correlations between a k -level reverse Horn fraction computed using an approximation of the solution of the linear program (instead of the optimal solution) and satisfiability and solver performance. On the other hand, one might try if it is possible to reduce the feasible region of the linear programs by using information that can be obtained by (fast) heuristics.

In this thesis, the new SOS (Sums Of Squares) approaches are compared with existing upper bound and rounding techniques, for the MAX-2-SAT case of Goemans and Williamson [60] and Feige and Goemans [53] and for the MAX-3-SAT case of

Karloff and Zwick [84], which are based on Semidefinite Programming as well. We prove or illustrate that for any of these algorithms there is a SOS-based counterpart providing upper bounds at least as tight, but observably tighter in particular cases.

We conclude that a combination of the rounding schemes of Goemans and Williamson and of Feige and Goemans with the appropriate SOS based upper bound techniques provides polynomial time algorithms for MAX-2-SAT having a performance guarantee at least as good as the ones proven by Goemans and Williamson and Feige and Goemans, but observably better in particular cases. A similar conclusion can be drawn with respect to the Karloff and Zwick algorithm for MAX-3-SAT.

In this thesis, several variants of a rounding procedure based on the SOS semidefinite program are presented. These different variants are compared experimentally. However, the scaling factors used by the rounding procedure seem a bit arbitrary and are mainly selected based on their experimental performance on the instances considered. It might be useful to try to find a more generic scaling factor, possibly as function of certain characteristics of the considered CNF-formula. A scaling factor depending on the characteristics of the instance might be useful because the relative performance of the different scaling factors studied seems to depend on the instance at hand. At this moment, the SOS-based method is too time-consuming for practical applications. However, it is expected that with a semidefinite programming solver that makes use efficiently of the sparseness and special structure of the SOS semidefinite programs, these could be solved much faster, possibly making the SOS-approach an attractive alternative for other MAXSAT-approximation algorithms for practical problems.

In this thesis, we presented a method to generate forbidden color instances based on Graph- k -SAT instances. On average, forbidden color instances are more difficult for state-of-the-art solvers than the Graph- k -SAT instances they originate from. The difficult satisfiability instances obtained in this way might be used for comparing the performance of satisfiability solvers. If forbidden color instances are generated randomly with an appropriate forbidden color percentage, a sufficient number of difficult instances is found. It is notable that a large number of the satisfiable forbidden color instances found are quite difficult to solve because usually, satisfiable graph- k -coloring instances are very easy to solve compared to unsatisfiable ones of the same size. Furthermore, our experiments show that structured problem instances have very different behaviour with respect to the forbidden color percentage. An analysis of the behavior of the instances with respect to the forbidden color percentage gives insight in the sensitivity of an instance for local disturbances. If a set of graph coloring instances coming from real-life scheduling or rostering problems would be available, it would be an interesting experiment to analyze their behavior with respect to the forbidden color percentage with a varying number of the different resources to get insight in the sensitivity of the problem for local disturbances for each of the resources. If an instance gets unsatisfiable after forbidding only a few colors, the instance is too sensitive for local disturbances. On the other hand, if many forbidden colors are needed to make the instance unsatisfiable, the set of resources is unnecessarily large and hence too expensive.

Bibliography

- [1] D. Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating satisfiable problem instances. In *Proceedings of the Seventeenth National Conference of Artificial Intelligence AAAI-2000*, pages 256–261, 2000.
- [2] F.A. Aloul, I.L. Markov, and K.A. Sakallah. Shatter: efficient symmetry-breaking for Boolean satisfiability. In *Proceedings of the Design Automation Conference*, pages 836–839, 2003.
- [3] T. Alsinet, F. Manya, and J. Planes. Improved branch and bound algorithms for max-sat. In *Proceedings of Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, 2003.
- [4] M.F. Anjos. On semidefinite programming relaxations for the satisfiability problem. *Mathematical Methods of Operations Research*, 60(3):349–367, 2004.
- [5] M.F. Anjos. An improved semidefinite programming relaxation for the satisfiability problem. *Mathematical Programming*, 102(3):589–608, 2005.
- [6] M.F. Anjos. Semidefinite optimization approaches for satisfiability and maximum-satisfiability problems. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:1–47, 2005.
- [7] T. Asano and D.P. Williamson. Improved approximation algorithms for MAXSAT. *Journal of Algorithms*, 42(1), 2002.
- [8] B. Aspvall, M.F. Plass, and R.E. Tarjan. A linear time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.
- [9] R. Beigel and D. Eppstein. 3-coloring in time $o(1.3289^n)$. *Journal of Algorithms*, 54(2):168–204, 2005.
- [10] H. Bennaceur and C.M. Li. Characterizing SAT problems with the row convexity property. In P. van Hentenrijk, editor, *Principles and Practice of Constraint Programming - CP2002*, LNCS 2470, pages 720–725, 2002.

- [11] E. Benoist and J.-J. Hebrard. Recognition of simple enlarged horn formulas and simple extended horn formulas. *Annals of Mathematics and Artificial Intelligence*, 37:251–272, 2003.
- [12] S.J. Benson, Y. Ye, and X. Zhang. Solving large-scale sparse semidefinite programs for combinatorial optimization. *SIAM Journal on Optimization*, 10(2):443–461, 2000.
- [13] S.J. Benson and Y.Ye. DSDP5 User guide - The dual-scaling algorithm for semidefinite programming. Technical Report ANL/MCS-TM-255, Argonne National Laboratory, 2005.
- [14] G. Biroli, S. Cocco, and R. Monasson. Phase transitions and complexity in computer science: an overview of the statistical physics approach to the random satisfiability problem. *Physica A*, 306:381–394, 2002.
- [15] G. Blekherman. There are significantly more nonnegative polynomials than sums of squares. submitted to Israel Journal of Mathematics, 2004.
- [16] A. Blum. New approximation algorithms for graph coloring. *Journal of the Association for Computing Machinery*, 41(3):470–516, 1994.
- [17] A. Blum and D. Karger. An $o(n^{3/14})$ -coloring for 3-colorable graphs. *Information Processing Letters*, 61(1):49–53, 1997.
- [18] B. Borchers. CSDP 4.9 user’s guide. <http://infohost.nmt.edu/~borchers/>.
- [19] B. Borchers. CSDP: A C library for semidefinite programming. *Optimization Methods and Software*, 11(1):613–623, 1999.
- [20] B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2(4):299–306, 1999.
- [21] E. Boros. Maximum renamable Horn sub-CNFs. *Discrete Applied Mathematics*, 96-97:29–40, 1999.
- [22] A. Braunstein, M. Mezard, and R. Zecchina. Survey Propagation: an algorithm for satisfiability. *Random Structures and Algorithms*, 27(2):201–226, 2005.
- [23] M. Cadoli and A. Schaerf. Compiling problem specifications into SAT. *Artificial Intelligence*, 162(1-2), 2005.
- [24] C. Castellini, E. Giunchiglia, and A. Tachella. SAT-based planning in complex domains: concurrency, constraints and non-determinism. *Artificial Intelligence*, 147:85–117, 2003.

- [25] V. Chandru and J.N. Hooker. Extended Horn sets in propositional logic. *Journal of the Association for Computing Machinery*, 38(1):205 – 221, 1991.
- [26] V. Chandru and J.N. Hooker. Detecting embedded Horn structure in propositional logic. *Information Processing Letters*, 42(2):109–111, 1992.
- [27] C.A. Chen and S.K. Gupta. A satisfiability-based test generator for path delay faults in combinational circuits. In *Proceedings of the 33rd Design Automation Conference*, 1996.
- [28] M.D. Choi, T.Y. Lam, and B. Reznick. Even symmetric sextics. *Mathematische Zeitschrift*, 195, 1987.
- [29] S.A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd annual ACM symposium on the Theory of Computing*, pages 151–158, 1971.
- [30] S.A. Cook and D.G. Mitchell. Finding hard instances of the satisfiability problem: a survey. In *DIMACS Series in Discrete mathematics and theoretical computer science*, volume 5. American Mathematical Society, 1997.
- [31] D. Coppersmith, D. Gamarnik, M.T. Hajiaghayi, and G.B. Sorkin. Random max sat, random max cut and their phase transitions. *Random Structures and Algorithms*, 24(4), 2004.
- [32] Y. Crama, O. Ekin, and P.L. Hammer. Variable and term removal from Boolean formulae. *Discrete Applied Mathematics*, 75(3):217–230, 1997.
- [33] J.M. Crawford and A.B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1994.
- [34] J. Culberson and I.P. Gent. Well out of reach: why hard problems are hard. Technical report apes-13-1999, APES Research Group, 1999.
- [35] E. Dantsin, M. Gavrilovich, E.A. Hirsch, and B. Konev. Max sat approximation beyond the limits of polynomial-time approximation. *Annals of Pure and Applied Logic*, 113(1-3):81–94, 2001.
- [36] G.B. Dantzig. Application of the simplex method to a transportation problem. In *Activity analysis of production and allocation - Proceedings of a Conference*, pages 359–373, 1951.
- [37] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the Association for Computing Machinery*, 5:394–397, 1962.

- [38] M. Davis and H. Putman. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.
- [39] E. de Klerk. *Aspects of Semidefinite Programming: Interior Point Algorithms and Selected Applications*, volume 65 of *Applied Optimization Series*. Kluwer Academic Publishers, 2002.
- [40] E. de Klerk and H. van Maaren. On semidefinite programming relaxations of (2+p)-SAT. *Annals of Mathematics and Artificial Intelligence*, 37(3):285–305, 2003.
- [41] E. de Klerk, H. van Maaren, and J.P. Warners. Relaxations of the satisfiability problem using semidefinite programming. *Journal of Automated Reasoning*, 24(1-2):37–65, 2000.
- [42] E. de Klerk and J.P. Warners. Semidefinite programming relaxations for MAX 2-SAT and 3-SAT: Computational perspectives. In *Combinatorial and Global Optimization, P.M. Pardalos, A. Migdalas, and R.E. Burkard (eds.), Series on Applied Optimization, Volume 14*. World Scientific Publishers, 2002.
- [43] A. del Val. On 2-SAT and renamable Horn. In *AAAI'00, Proceedings of the Seventeenth (U.S.) National Conference on Artificial Intelligence*, pages 279–284, 2000.
- [44] D.D. Demopoulos and M.Y. Vardi. The phase transition in the random 1-3-hornsat problem, 2004.
- [45] G. Dequen and O. Dubois. kcnfs: An efficient solver for random k-sat formulae. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003*, Lecture Notes in Computer Science 2919, pages 486–501, 2003.
- [46] P. Di Giacomo, G. Felici, R. Maceratini, and K. Truemper. Application of a new logic domain method for the diagnosis of hepatocellular carcinoma. In *Proceedings of the Tenth World Congress on Health and Medical Informatics*, 2001.
- [47] M.B. Do and S. Kambhampati. On the use of LP relaxations in solving SAT encodings of planning problems. In *Working note of the Workshop on the Integration of AI and OR Techniques for Combinatorial Optimization, AAAI-2000*, 2000.
- [48] W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.
- [49] M.R. Dransfield, L. Liu, V.W. Marek, and M. Truszczynski. Satisfiability and computing van der Waerden numbers. *The Electronic Journal of Combinatorics*, 11(1), 2004.

- [50] P.E. Dunne and T.J.M. Bench-Capon. A sharp threshold for the phase transition of a restricted satisfiability problem for Horn clauses. *Journal of Logic and Algebraic Programming*, 47(1):1–14, 2001.
- [51] T. Erlebach, A. Hall, and T. Schank. Classifying customer-provider relationships in the internet. In *Proceedings of the IASTED International Conference on Communications and Computer Networks (CCN 2002)*, pages 538–545, 2002.
- [52] M. Ernst, T.D. Millstein, and D.S. Weld. Automatic SAT-compilation of planning problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1169–1177, 1997.
- [53] U. Feige and M.X. Goemans. Approximating the value of two prover proof systems, with applications to MAX2SAT and MAXDICUT. In *Proceedings of the Third Israel Symposium on Theory of Computing and Systems*, pages 182–189, 1995.
- [54] C. Fiorini, E. Martinelli, and F. Massacci. How to take an RSA signature by encoding modular root finding as a SAT problem. *Discrete Applied Mathematics*, 130:101–127, 2003.
- [55] J. Franco. Results related to threshold phenomena research in satisfiability: lower bounds. *Theoretical Computer Science*, 265(1-2):147 – 157, 2001.
- [56] E. Friedgut. Sharp thresholds of graph properties and the k -SAT problem. *Journal of the American Mathematical Society*, 12(4):1017–1054, 1999.
- [57] K. Fujisawa, M. Kojima, and K. Nakata. Exploiting sparsity in primal-dual interior-point methods for semidefinite programming. *Mathematical Programming*, 79:235–253, 1997.
- [58] K. Fujisawa, M. Kojima, K. Nakata, and M. Yamashita. SDPA (Semidefinite Programming Algorithm) : user’s manual, version 6.00. Research Reports on Mathematical and Computing Sciences, Series B : Operations Research B308, Department of Mathematical and Computing Science, Tokyo Institute of Technology, 2002.
- [59] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified \mathcal{NP} -complete graph problems. *Theory of Computer Science*, 1:237–267, 1976.
- [60] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association of Computing Machinery*, 42(6):1115–1145, 1995.
- [61] A. Goerdt. A threshold for unsatisfiability. *Journal of computer and system Sciences*, 53(3):469–486, 1996.

- [62] E. Goldberg and Y. Novikov. Berkmin: a fast and robust sat-solver. In *Proceedings of the Design Automation and Test in Europe (DATE-2002)*, pages 142–149, 2002.
- [63] C.P. Gomes and D. Shmoys. Completing quasigroups or latin squares: a structured graph coloring problem. In *Proceedings of the Computational Symposium on Graph Coloring and Generalizations*, 2002.
- [64] L. Guerra e Silva, J. Marques-Silva, and L.M. Silveira. Satisfiability models and algorithms for circuit delay computation. *ACM Transactions on Design Automation of Electronic Systems*, 7(1):137–158, 2002.
- [65] V. Guruswami and S. Khanna. On the hardness of 4-coloring a 3-colorable graph. *SIAM Journal on Discrete Mathematics*, 18(1):30–40, 2004.
- [66] E. Halperin and U. Zwick. Approximation algorithms for max-4-sat and rounding procedures for semidefinite programs. *Journal of Algorithms*, 40:184–211, 2001.
- [67] J. Håstad. Some optimal inapproximability results. *Journal of the Association of Computing Machinery*, 48:798–859, 2001.
- [68] C. Helmberg, F. Rendl, R.J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM Journal in Optimization*, 6(2):342–361, 1996.
- [69] D. Henrion and J.-B. Lasserre. Gloptipoly: global optimization over polynomials with MATLAB and SeDuMi. *ACM Transactions on Mathematical Software*, 29(2):165–194, 2003.
- [70] D. Henrion and J.-B. Lasserre. Detecting global optimality and extracting solutions in GloptiPoly. In D. Henrion and A. Garulli, editors, *Positive polynomials in control. Lecture Notes on Control and Information Sciences*. Springer Verlag, 2005.
- [71] P.R. Herwig, M.J.H. Heule, P.M. Van Lambalgen, and H. Van Maaren. A new method to construct lower bounds for van der waerden numbers. *The Electronic Journal of Combinatorics*, 12, 2005.
- [72] M. Heule, M. Dufour, J. van Zwieten, and H. van Maaren. March_eq, implementing additional reasoning into an efficient look-ahead sat-solver. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing*, number 3542 in Springer Lecture Notes on Computer Science, pages 345–359, 2005.
- [73] F.S. Hillier and G.J. Lieberman. *Introduction to operations research*. McGrawHill, 8th edition edition, 2005.

- [74] E. A. Hirsch and A. Kojevnikov. UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. *Annals of Mathematics and Artificial Intelligence*, 43(1-4):91–111, 2005.
- [75] J. Hooker. *Logic-based methods for optimizations, combining optimization and constraint satisfaction*. Wiley, 2000.
- [76] H.H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proceedings of the Eighteenth national conference on Artificial intelligence (AAAI-02)*, pages 655–660, 2002.
- [77] S. Hossain and T. Steihaug. Graph coloring in the estimation of mathematical derivatives. In *Graph Coloring and its Generalizations (Coloring02)*, 2002.
- [78] S. Hossain and T. Steihaug. Optimal direct determination of sparse Jacobian matrices. Technical Report 254, Department of Informatics, University of Bergen, Norway, 2003.
- [79] Inc. ILOG. *ILOG CPLEX 7.1 Reference Manual*. 2001.
- [80] G. Istrate. The phase transition in random Horn satisfiability and its algorithmic implications. *Random Structures and Algorithms*, 20(4):483–506, 2002.
- [81] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part II, graph coloring and number partitioning. *Operations Research*, 31:378–406, 1991.
- [82] S. Joy, J. Mitchell, and B. Borchers. A branch and cut algorithm for max-sat and weighted max-sat. In D. Du, J. Gu, and P.M. Pardalos, editors, *DIMACS series in discrete mathematics and theoretical computer science*, volume 35. American Mathematical Society, 1997.
- [83] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *Journal of the Association of Computing Machinery*, 45(2):246 – 265, 1998.
- [84] H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX 3SAT? In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*. IEEE Press, 1997.
- [85] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.
- [86] H. Kautz. Satplan04: Planning as satisfiability. In *ICAPS'04*, 2004.
- [87] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning (KR'96)*, 1996.

- [88] H. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, 1992.
- [89] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, 1996.
- [90] D.E. Knuth. *The art of computer programming, vol. 2: seminumerical algorithms*. Addison-Wesley, 1969.
- [91] M. Krivelevich and B. Sudakov. Coloring random graphs. *Information Processing Letters*, 67(2):71–74, 1998.
- [92] O. Kullmann. First report on an adaptive density based branching rule for DLL-like SAT-solvers, using a database of mixed random conjunctive normal forms created using the advanced encryption standard (aes). Computer Science Report Series CSR 19-2002, University of Wales Swansea, 2002.
- [93] O. Kullmann. Investigating the behavior of a SAT solver on random formulas. Submitted to *Annals of Mathematics and Artificial Intelligence*, 2002.
- [94] M. Laurent and F. Rendl. Semidefinite programming and integer programming. To appear as chapter of the *Handbook on Discrete Optimization* edited by K.Aardal, G. Nemhauser and R. Weismantel.
- [95] M. Lewin, D. Livnat, and U. Zwick. Improved rounding techniques for the max-2-sat and max-di-cut problems. In *Proceedings of 9th Integer Programming and Combinatorial Optimization Conference*, LNCS 2337, pages 67–82, 2002.
- [96] C.M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability. In *International Joint Conference on Artificial Intelligence*, pages 366–371, 1997.
- [97] C.M. Li, B. Jurkowiak, and P.W. Purdom. Integrating symmetry breaking into a DLL procedure. In *Proceedings of the International Symposium on Boolean Satisfiability (SAT), Cincinnati*, pages 149–155, 2002.
- [98] I. Lynce. *Propositional satisfiability: techniques, algorithms and applications*. PhD thesis, Universidade Tecnica de Lisboa, 2004.
- [99] I. Lynce and J.P. Marques-Silva. Efficient data structures for backtrack search SAT solvers. *Annals of Mathematics and Artificial Intelligence*, 43(1-4):137–152, 2005.
- [100] S. Mahajan and H. Ramesh. Derandomizing approximation algorithms based on semidefinite programming. *SIAM Journal on Computing*, 28(5):1641–1663, 1999.

- [101] V.M. Manquinho, J.P. Marques Silva, A.L. Oliveira, and K.A. Sakallah. Satisfiability-based algorithms for 0-1 integer programming. In *International Workshop on Logic Synthesis*, 1998.
- [102] A. Marino and R.I. Damper. Breaking the symmetry of the graph coloring problem with genetic algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 240–245, 2000.
- [103] J.P. Marques Silva and K.A. Sakallah. Robust search algorithms for test pattern generation. In *Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)*, 1997.
- [104] F. Massacci and L. Marraro. Logical cryptanalysis as a SAT problem: encoding and analysis of the U.S. Data Encryption Standard. *Journal of Automated Reasoning*, 24:165–203, 2000.
- [105] M. Mastrolilli and L.M. Gambardella. Maximum satisfiability: How good are tabu search and plateau moves in the worst-case? *European Journal of Operational Research*, 166:63–76, 2005.
- [106] S. Matuura and T. Matsui. New approximation algorithms for max-2-sat and max-dicut. *Journal of the Operations Research Society of Japan*, 46(2):178–188, 2003.
- [107] R. Monasson. On the analysis of backtrack procedures for the coloring of random graphs. In E. Ben-Naim, H. Frauenfelder, and Z. Torczkai, editors, *Complex Networks*. Springer-Verlag, 2004.
- [108] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400:133–137, 1999.
- [109] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the 39th Design Automation Conference, Las Vegas*, 2001.
- [110] Y.E. Nesterov and A. Nemirovskii. *Interior-point polynomial algorithms in convex programming*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1994.
- [111] N. Nishimura, P. Ragde, and S. Szeider. Detecting backdoor sets with respect to Horn and binary clauses. In *submitted to Discrete Applied Mathematics*, 2004.
- [112] P.A. Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, California Institute of Technology, Pasadena, CA, 2000.

- [113] P.A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming Series B*, 96(2):293–320, 2003.
- [114] P.A. Parrilo and B. Sturmfels. Minimizing polynomial functions. In *Algorithmic and quantitative real algebraic geometry ,DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 60*, pages 83–99. American Mathematical Society, 2001.
- [115] V. Th. Paschos. Polynomial approximation and graph-coloring. *Computing*, 70, 2003.
- [116] S. Porschen and E. Speckenmeyer. Worst case bounds for some \mathcal{NP} -complete modified Horn-SAT problems. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing*, Lecture Notes in Computer Science, no. 3542, pages 251–262, 2005.
- [117] S. Porschen, E. Speckenmeyer, B. Randerath, and M. Gärtner. Tabu-sat and walksat for level graph formula. Technical Report Technical Report: zaik2004-476, Zentrum für Angewandte Informatik Köln, 2004.
- [118] N.R. Potlapally. Solving register allocation using boolean satisfiability.
- [119] S. Prajna, A. Papachristodoulou, P. Seiler, and P.A. Parrilo. Sostools, sum of squares optimization toolbox for matlab. Available from <http://www.cds.caltech.edu/sostools>, June 2004.
- [120] M. Putinar. Positive polynomials on compact semi-algebraic sets. *Indiana University Mathematics Journal*, 42(3), 1993.
- [121] J. Rintanen. Evaluation strategies for planning as satisfiability. In *Proceedings of the European Conference on Artificial Intelligence*, pages 682–687, 2004.
- [122] J. Rintanen. Phase transitions in classical planning: an experimental study. In *Proceedings of ICAPS 2004*, pages 101–110, 2004.
- [123] J. Rintanen, K. Heljanko, and I. Niemela. Parallel encodings of classical planning as satisfiability. In J.J. Alferes and J. Leite, editors, *Logics in Artificial Intelligence: 9th European Conference, JELIA 2004, Lecture Notes in Computer Science, volume 3229*, pages 307–319. Springer-Verlag, 2004.
- [124] C. Roos, T. Terlaky, and J.-Ph. Vial. *Theory and algorithms for linear optimization*. Wiley, 1997.
- [125] A. Sabharwal. Symchaff: A structure-aware satisfiability solver. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, 2005.

- [126] J.A. Schlipf, F.S. Annexstein, J.V. Franco, and Swaminathan. On finding solutions for extended horn formulas. *Information Processing Letters*, 54:133–137, 1995.
- [127] B. Selman and H. Kautz. Knowledge compilation using horn approximations. In *Proceedings of the AAAI-91*, pages 904–909, 1991.
- [128] B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the AAAI-94*, 1994.
- [129] B. Selman, H.J. Levesque, and D.G. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence ,San Jose, CA*, pages 440–446, 1992.
- [130] H. Shen and H. Zhang. An empirical study of max-2-sat phase transitions. In *Proceedings of LICS'03 Workshop on Typical Case Complexity and Phase Transitions*, 2003.
- [131] H. Shen and H. Zhang. Improving exact algorithms for max-2-sat. *Annals of Mathematics and Artificial Intelligence*, 44(4):419–43, 2005.
- [132] A. Smith, A. Veneris, M.F. Ali, and A. Viglas. Fault diagnosis and logic debugging using boolean satisfiability. *IEEE Transactions on COMPUTER-AIDED DESIGN of Integrated Circuits and Systems*, 24(9), 2005.
- [133] I. Spence. The sudoku puzzle as a satisfiability problem. <http://www.cs.qub.ac.uk/I.Spence/SuDoku/SuDoku.html>.
- [134] J.F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999.
- [135] T. Stützle, H.H. Hoos, and A. Roli. A review of the literature on local search algorithms for max-sat. Technical report aida-01-02, FG Intellektik, FB Informatik, TU Darmstadt, Germany, 2001.
- [136] P. Svenson and M.G. Nordahl. Relaxation in graph coloring and satisfiability problems. *Physical Review E*, 59(4):3883–3999, 1999.
- [137] D.A.D. Tompkins and H.H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing*, Springer Lecture Notes on Computer Science, no. 3542, 2005.
- [138] L. Trevisan, G.B. Sorkin, M. Sudan, and D.P. Williamson. Gadgets, approximation, and Linear Programming. *SIAM Journal on Computing*, 29(6):2074–2097, 2000.

- [139] A. Van Gelder. Generalizations of watched literals for backtracking. In *Proceedings of the Seventh International Symposium on Artificial Intelligence and Mathematics*, 2001.
- [140] H. van Maaren and C. Dang. Simplicial pivoting algorithms for a tractable class of integer programs. *Journal of Combinatorial Optimization*, 6(2):133–142, 2002.
- [141] H. van Maaren and L. van Norden. Hidden threshold phenomena for fixed-density SAT-formulae. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003, Selected Revised Papers, Lecture Notes of Computer Science, LNCS 2919*, pages 135–149, 2004.
- [142] H. van Maaren and L. van Norden. Correlations between Horn fractions, satisfiability and solver performance for fixed density random 3-CNF instances. *Annals of Mathematics and Artificial Intelligence*, 44:157–177, 2005.
- [143] H. van Maaren and L. van Norden. Sums of squares based approximation algorithms for max-sat. *submitted to Discrete Applied Mathematics*, 2005.
- [144] H. van Maaren and L. van Norden. Sums of squares, satisfiability and maximum satisfiability. In F. Bacchus and T. Walsh, editors, *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing*, volume Lecture Notes in Computer Science 3569, pages 293–307, 2005.
- [145] H. van Maaren and J.P. Warners. Solving satisfiability problems using elliptic approximations. A note on volumes and weights. *Annals of Mathematics and Artificial Intelligence*, 37(3):273–283, 2003.
- [146] L. van Norden and S.L. van de Velde. Multi-product lot-sizing with a transportation capacity reservation contract. *European Journal of Operational Research*, 165(1):127–138, 2005.
- [147] L. van Norden and H. van Maaren. A linear programming based satisfiability solver using a new Horn-driven search tree design. In P. van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP2002*, Lecture Notes in Computer Science 2470, pages 775–776. Springer, September 2002.
- [148] L. van Norden, J.A.E.E. van Nunen, and S.L. van de Velde. A winner determination problem of tendering transportation services. *Zeitschrift für Betriebswirtschaft*, 2006.
- [149] H. Waki, S. Kim, M. Kojima, and M. Muramatsu. Sums of squares and semidefinite programming relaxations for polynomial optimization problems with structured sparsity. Research reports on Mathematical and Computing Sciences B-411, Tokyo Institute of Technology, Department of Mathematical and Computing Sciences, 2004, revised 2005.

- [150] J.P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998.
- [151] S.A. Wolfman and D.S. Weld. Combining linear programming and satisfiability solving for resource planning. *The Knowledge Engineering Review*, 16(1):85–99, 2001.
- [152] K. Xu and W. Li. Many hard examples in exact phase transitions with application to generating hard satisfiable instances. CoRR Report cs.CC/0302001, 2003.
- [153] M. Yamashita, K. Fujisawa, and M. Kojima. Implementation and evaluation of SDPA 6.0. Research Reports on Mathematical and Computing Sciences, Series B : Operations Research B383, Department of Mathematical and Computing Science, Tokyo Institute of Technology, 2002.
- [154] H. Zhang and M.E. Stickel. An efficient algorithm for unit propagation. In *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI-MATH'96)*, 1996.
- [155] H. Zhang and M.E. Stickel. Implementing the Davis-Putnam procedure. *Journal of Automated Reasoning*, 24(1/2):277–296, 2000.
- [156] U. Zwick. Computer assisted proof of optimal approximability results. In *Proceedings of 13th SODA*, pages 496–505, 2002.

Summary

The satisfiability problem is an important problem in theoretical computer science because its central position in complexity theory. Some well-known \mathcal{NP} -complete problems like graph coloring and some planning problems can be translated into the satisfiability problem in an efficient way. Satisfiability solving procedures are becoming interesting alternatives for an increasing number of problem domains.

Special cases of the satisfiability problem that can be solved in polynomial time attracted quite some attention. Two of the best-known examples include 2-SAT and Horn satisfiability. A Horn clause is a clause containing at most one positive literal. Reverse Horn clauses, i.e. clauses containing at most one negative literal, play a central role in this thesis. The reverse Horn fraction equals the number of reverse Horn clauses divided by the total number of clauses.

It is well-known that the k -SAT problem, with only clauses with at most k literals, shows a phase transition with respect to the clause-variable density. The density of a CNF-formula is the number of clauses divided by the number of variables. With increasing density the fraction of satisfiable instances among a set of randomly generated instances decreases. This thesis contains experimental evidence for the fact that $\{3\}$ -SAT instances with a fixed clause-variable density show another phase transition with respect to an enhanced Horn fraction. This enhanced Horn fraction implies averaging an approximation of the optimal reverse Horn fractions in a selected set of nodes in a search tree specially designed for this purpose. This enhanced Horn fraction is correlated with the performance of complete and incomplete satisfiability solvers and with the satisfiability of an instance as can be concluded from the experimental results. The correlations between enhanced Horn fraction, satisfiability and solver performance are not only present for the $\{3\}$ -SAT problem but are even more prominent for graph-3-coloring instances after translation into satisfiability instances.

A method based on semidefinite programming for computing upper and lower bounds for the MAX-SAT problem is presented. MAX-SAT is an optimization variant of the satisfiability problem that asks for the maximum number of clauses that can be satisfied. Goemans and Williamson present a semidefinite programming based algorithm for computing a 0.878-approximation for MAX-2-SAT. The new approach presented in this thesis for approximating MAX-SAT uses a semidefinite program approximating a function expressing the maximum number of clauses that can be satisfied. The minimum ϵ of a polynomial $F(x_1, \dots, x_n)$ can be approximated by the maximal ϵ such that $F(x_1, \dots, x_n) - \epsilon$ is a sum of squares, because every sum of squares

is clearly non-negative but not every non-negative polynomial is a sum of squares. The problem of finding the maximal ϵ such that $F(x_1, \dots, x_n) - \epsilon$ is a sum of squares can be solved in polynomial time up to a prescribed precision by semidefinite programming. In this way, an upper bound for the MAX-SAT solution can be obtained. In this approach a basis of monomials has to be chosen with respect to which the semidefinite program is solved.

We prove that the semidefinite programs from the approach of Goemans and Williamson and the Sums of Squares (SOS) approach with a monomial basis containing only monomials of degree at most one are dual semidefinite programs and hence give the same upper bounds for MAX-2-SAT. Adding monomials of degree two to the monomial basis in the SOS-approach is proved to give upper bounds that are at least as tight as when the corresponding valid inequalities of the type proposed by Feige and Goemans are added to the semidefinite program of Goemans and Williamson. Experiments show that for sufficiently large random 3-SAT instances the SOS-approach with a monomial basis containing the appropriate monomials of degree at most three is capable in proving unsatisfiability for more instances than the rank-3-relaxation of Anjos.

Besides the semidefinite program for computing upper bounds for MAX-SAT, a rounding procedure is presented for computing lower bounds based on the solution of the semidefinite program. The observed performance guarantee of an algorithm equals the average lower bound over a number of runs divided by the realized upper bound. The experiments illustrate that the observed performance guarantee of the SOS-approach with a monomial basis containing the appropriate monomials of degree at most two is better than the observed performance guarantee of the algorithm by Goemans and Williamson and the proven performance guarantee of the algorithm of Lewin, Livnat and Zwick which has the best proven performance guarantee till now. For a set of random 3-SAT instances the observed performance guarantee of the SOS-approach with a monomial basis containing monomials of degree at most three is better than the observed performance guarantee of the method of Karloff and Zwick.

For testing and comparing solvers for the satisfiability problem it is important to have hard instances available. Graph- k -coloring problems tend to be more difficult in the neighborhood of the phase transition with respect to the node-edge density. This thesis presents a method to generate satisfiability instances coming from graph- k -coloring that are experimentally shown to be more difficult than the instances they are based on. The idea of these so-called forbidden color instances is adding unit literal clauses to the CNF-formula that express that certain nodes are not allowed to have certain colors. Studying the behavior of structured instances with respect to the forbidden color percentage gives insight in the sensitivity of the instances for local disturbances.

Samenvatting

Het satisfiability probleem is een belangrijk probleem uit de theoretische informatica vanwege zijn centrale positie in de complexiteitstheorie. Sommige bekende \mathcal{NP} -moeilijke problemen zoals graph coloring en sommige planningsproblemen kunnen vertaald worden naar het satisfiability probleem op een efficiënte manier. Methodes om het satisfiability probleem op te lossen worden interessante alternatieven voor een toenemend aantal probleemgebieden. Speciale gevallen van het satisfiability probleem die in polynomiale tijd opgelost kunnen worden hebben aardig wat aandacht getrokken. De twee bekendste voorbeelden zijn 2-SAT en Horn satisfiability. Een Horn clause is een clause die maximaal één positieve literal bevat. Reverse Horn clauses, d.w.z. clauses die maximaal één negatieve literal bevatten, spelen een centrale rol in dit proefschrift. De reverse Horn fractie is gelijk aan het aantal reverse Horn clauses gedeeld door het totaal aantal clauses.

Het is bekend dat het k -SAT probleem, met alleen clauses met maximaal k literals, een faseovergang laat zien met betrekking tot de clause-variabele dichtheid. De dichtheid van een CNF-formule is het aantal clauses gedeeld door het aantal variabelen. Met toenemende dichtheid neemt de fractie satisfiable instanties in een verzameling random gegenereerde instanties af. Dit proefschrift geeft experimenteel bewijs voor het feit dat $\{3\}$ -SAT instanties met een vaste clause-variabele dichtheid een andere faseovergang laten zien met betrekking tot een enhanced Horn fractie. Deze enhanced Horn fractie impliceert het middelen van een benadering van de optimale reverse Horn fracties in een geselecteerde verzameling nodes in een zoekboom die speciaal voor dit doel is ontworpen. Deze enhanced Horn fractie is gecorreleerd met de prestaties van volledige en onvolledige satisfiability solvers en met de satisfiability van een instantie zoals volgt uit de experimentele resultaten. De correlaties tussen enhanced Horn fractie, satisfiability en de prestaties van solvers zijn niet alleen aanwezig voor het $\{3\}$ -SAT probleem maar zijn zelfs duidelijker voor graph-3-coloring instanties na vertaling naar satisfiability instanties.

Een op semidefiniet programmeren gebaseerde methode om boven- en ondergrenzen voor het MAX-SAT-probleem te berekenen wordt beschreven. MAX-SAT is een optimalisatie variant van het satisfiability probleem die vraagt naar het maximale aantal clauses waar aan voldaan kan worden. Goemans en Williamson beschrijven een op semidefiniet programmeren gebaseerd algoritme om een 0.878-benadering voor MAX-2-SAT te berekenen. De nieuwe methode uit dit proefschrift om MAX-SAT te benaderen gebruikt een semidefiniet programma dat een functie benadert die het

maximaal aantal clauses aangeeft waar aan voldaan kan worden. Het minimum ϵ van een polynoom $F(x_1, \dots, x_n)$ kan benaderd worden door de maximale ϵ zodat $F(x_1, \dots, x_n) - \epsilon$ een som van kwadraten is, omdat elke som van kwadraten duidelijk niet-negatief is maar niet elk niet-negatief polynoom een som van kwadraten is. Het probleem om de maximale ϵ te vinden zodat $F(x_1, \dots, x_n) - \epsilon$ een som van kwadraten is kan in polynomiale tijd opgelost worden op een voorgeschreven nauwkeurigheid na met behulp van semidefinitief programmeren. Op deze manier kan een bovengrens voor de MAX-SAT oplossing worden gevonden. In deze methode moet een monomial basis worden gekozen waarmee het semidefinitief programma wordt opgelost. We bewijzen dat de semidefinitieve programma's in de aanpak van Goemans en Williamson en de Sums of Squares (SOS) aanpak met een monomial basis met alleen monomials van graad maximaal één duale semidefinitieve programma's zijn en daardoor dezelfde bovengrenzen geven voor MAX-2-SAT. Het is bewezen dat het toevoegen van monomials van graad twee aan de monomial basis in de SOS-aanpak bovengrenzen geeft die minstens zo goed zijn als wanneer de corresponderende valid inequalities van het type zoals voorgesteld door Feige en Goemans worden toegevoegd aan het semidefinitieve programma van Goemans en Williamson. Experimenten laten zien dat voor voldoende grote random 3-SAT instanties, de SOS-aanpak met een monomial basis met monomials van graad maximaal drie in staat is om de unsatisfiability te bewijzen van meer instanties dan de rank-3-relaxatie van Anjos. Behalve het semidefinitieve programma om bovengrenzen voor MAX-SAT te berekenen, wordt een afrondingsprocedure beschreven om ondergrenzen te berekenen gebaseerd op de oplossing van het semidefinitieve programma. De observed performance guarantee van een algoritme is gelijk aan de gemiddelde ondergrens over een aantal pogingen gedeeld door de gerealiseerde bovengrens. De experimenten illustreren dat de observed performance guarantee van de SOS-aanpak met een monomial basis met de geschikte monomials van graad maximaal twee beter is dan de observed performance guarantee van het algoritme van Goemans en Williamson en de bewezen performance guarantee van het algoritme van Lewin, Livnat en Zwick dat tot op heden de beste bewezen performance guarantee heeft. Voor een verzameling random 3-SAT instanties is de observed performance guarantee van de SOS-aanpak met een monomial basis met monomials van graad maximaal drie beter dan de observed performance guarantee van de methode van Karloff en Zwick.

Om solvers voor het satisfiability probleem te testen en te vergelijken is het belangrijk om over moeilijke instanties te beschikken. Graph- k -coloring problemen hebben de neiging moeilijker te zijn in de buurt van de faseovergang met betrekking tot de node-edge dichtheid. Dit proefschrift beschrijft een methode om satisfiability instanties afkomstig van graph- k -coloring te genereren waarvan experimenteel is aangetoond dat ze moeilijker zijn dan de graph- k -coloring instanties waar ze vandaan komen. Het idee van deze zogenaamde forbidden color instanties is dat unit literal clauses worden toegevoegd aan de CNF-formule die aangeven dat sommige nodes bepaalde kleuren niet mogen krijgen. Het bestuderen van het gedrag van gestructureerde instanties met betrekking tot het forbidden color percentage geeft inzicht in de gevoeligheid van de instanties voor locale verstoringen.